PETRE PREOTEASA
LUCA-DAN ȘERBĂNAȚI

MATEMATICĂ APLICATĂ ÎN TEHNICA DE CALCUL

XI

MINISTERUL EDUCAȚIEI ȘI ÎNVĂȚĂMÎNTULUI

PETRE PREOTEASA
LUCA – DAN ȘERBĂNAȚI

MATEMATICĂ APLICATĂ ÎN TEHNICA DE CALCUL

XI

Manual pentru clasa a XI-a
licee de matematică-fizică



EDITURA DIDACTICĂ ȘI PEDAGOGICĂ
BUCUREȘTI

Capitolul I

ELEMENTE DE BAZĂ ALE PROGRAMĂRII CALCULATOARELOR

Limbajele de programare sînt mijloace de comunicare ale omului cu calculatorul. Ele sînt inventate de om și servesc utilizatorilor pentru a exprima ceea ce trebuie să execute calculatorul.

Se știe că un calculator este un obiect inert atîta vreme cît nu i se indică operațiile pe care trebuie să le execute. Aceste operații sînt exprimate de către utilizator sub forma unor comenzi sau indicații și transmise astfel calculatorului.

> Succesiunea de comenzi sau indicații transmise calculatorului în vederea executării lor se numește *program*. O comandă sau indicație dintr-un program se numește *instrucțiune*.

Exprimînd într-un program *ceea ce trebuie să facă* un calculator, utilizatorul îi transmite de fapt un algoritm. Limbajul de programare ne dă tocmai modul în care descriem acest algoritm. O definiție echivalentă a programului va fi deci următoarea :

> Un program este descrierea unui algoritm într-un limbaj de programare.

Algoritmul, concept abstract, capătă prin program o reprezentare simbolică. După cum se știe, o altă modalitate de reprezentare (descriere) simbolică a unui algoritm este schema logică.

În primul capitol al manualului vom introduce limbajul de programare simplu LPS, pe baza căruia vom explica concepte și noțiuni importante în programarea calculatoarelor.

Proiectarea LPS a avut în vedere o familie de limbaje folosite în prezent de majoritatea programatorilor : limbajele de tip *pseudocod*. *Ele nu sînt în realitate limbaje de programare*, ci folosesc la descrierea algoritmilor într-o formă asemănătoare limbajului natural. Algoritmii descriși în pseudocod pot fi apoi cu ușurință transcriși într-un limbaj de programare (ALGOL, PASCAL, FORTRAN), transcrierea fiind aproape mecanică dacă se cunosc cele cîteva reguli necesare.

Prezentarea limbajului LPS se va face prin intermediul sistemului de prelucrare a datelor simplu.

I.1. SISTEMUL DE PRELUCRARE A DATELOR SIMPLU (SPDS)

Vom prezenta în cele ce urmează un sistem de calcul cu o funcționare foarte simplă, sistem pe care îl vom numi pe scurt SPDS. *El nu există în realitate*, scopul prezentării fiind în primul rînd didactic : plecînd de la funcționarea lui vom introduce notiuni şi concepte de bază în programarea sistemelor de calcul reale.

Structura şi funcţionarea SPDS au rezultat prin simplificări repetate ale structurii şi funcţionării sistemelor reale de prelucrare a datelor, reținînd de la acestea numai aspectele importante pentru utilizator. Deci putem afirma :

> Sistemul de prelucrare a datelor simplu (SPDS) este un *model* (o imagine simplificată) al sistemelor de calcul reale.

I.1.1. Structura şi funcţionarea SPDS

1. Structura SPDS este cea din figura I.1.

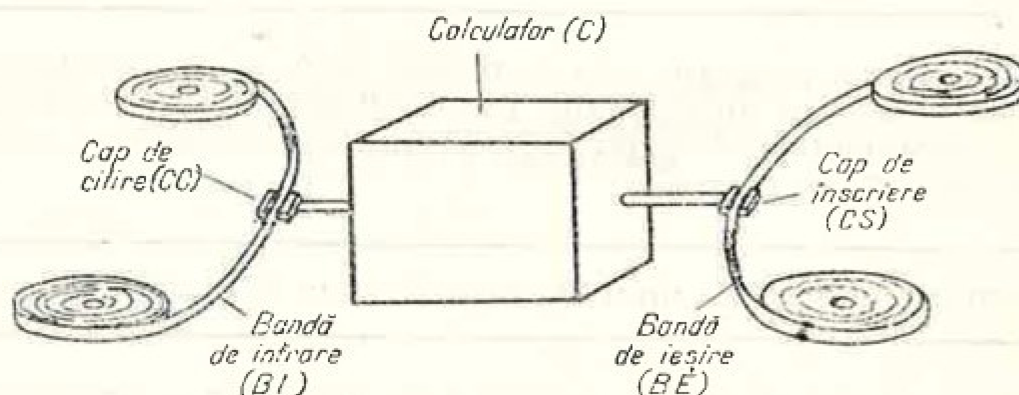


Fig. I.1

> SPDS este alcătuit din :
> — banda de intrare BI,
> — banda de ieşire BE şi
> — calculatorul C.

Vom descrie în continuare părțile componente ale SPDS şi apoi funcționarea întregului ansamblu.

2. *Benzile de intrare și ieșire* (BI, respectiv BE) servesc la citirea datelor inițiale în vederea prelucrării lor de către calculatorul C și respectiv la înscrierea rezultatelor calculelor realizate de C.

> BI și BE sînt modele ale fișierelor de intrare (pachetul de cartele perforate conținînd datele inițiale) respectiv de ieșire (succesiunea de linii scrise la imprimantă) ale sistemelor de prelucrare automată a datelor.

Succesiunea datelor conținute pe fiecare din cele două benzi va forma și în acest caz un fișier, *fișierul de intrare* și respectiv *fișierul de ieșire* ale SPDS.

Putem să ne facem o idee asupra acestor benzi gîndindu-ne la banda magnetofonului sau la banda de hîrtie a telegrafului. În plus, benzile BI și BE sînt divizate în celule, fiecare celulă conținînd o informație (dată) „depusă" în prealabil printr-un procedeu de înregistrare magnetică, tipărire etc. De altfel, trebuie spus că nu ne interesează în legătură cu aceste benzi natura suportului informației și nici modul de reprezentare a datelor, ci numai interpretarea lor. În acest sens:

> Restrîngem interpretarea datelor înscrise pe benzi la următoarele tipuri:
> — numere întregi,
> — numere reale,
> — valori logice,
> — șiruri de caractere, numite și texte (le vom prezenta cuprinse între ghilimele sau apostrofuri).

În figura I.2 sînt prezentate cîteva exemple de tipuri de date.

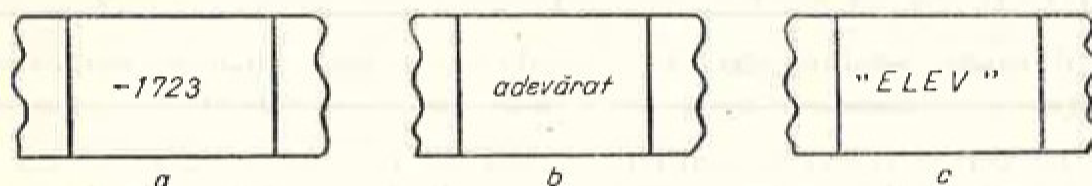


Fig. I.2

În cazul în care o celulă nu conține nici o dată din tipurile de date amintite, considerăm — prin convenție — că ea conține un caracter special numit „spațiu" sau „blanc" notat de obicei cu ␣.

3.

> *Calculatorul C* are următoarele părți componente:
> — capul de citire CC,
> — capul de înscriere CS,
> — memoria M,
> — dispozitivul de prelucrare DP și
> — dispozitivul de comandă DC.

Schema-bloc a calculatorului C este prezentată în figura I.3.

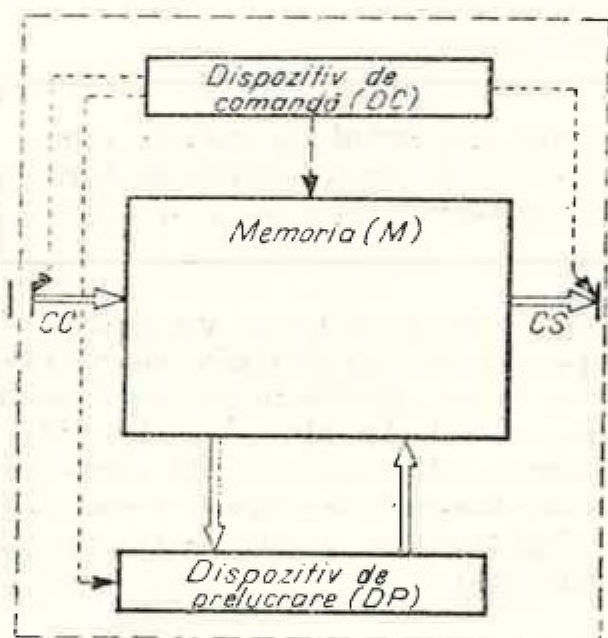


Fig. I.3

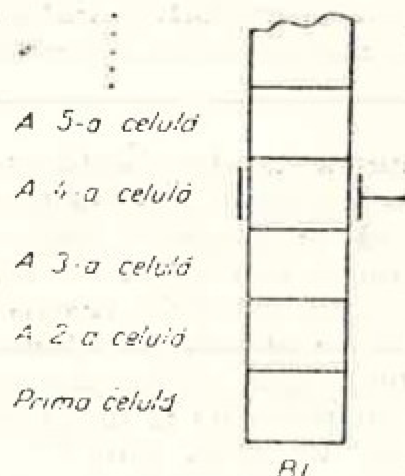


Fig. I.4

Schema arată, în afara componentelor amintite, transferul datelor între componente cu săgeată dublă și transferul comenzilor cu săgeată punctată. Vom detalia în continuare componentele lui C.

4. *Capul de citire CC* (asemănător celui de la magnetofon, de exemplu) folosește la citirea datelor aflate pe banda BI. Pentru aceasta CC este întotdeauna poziționat exact pe o celulă a benzii numită *celulă curentă*. După citirea datei aflată în celula curentă, banda se deplasează într-un singur sens astfel încît sub capul de citire să ajungă celula următoare care devine noua celulă curentă.

Avansul benzii se face sacadat, celulă cu celulă, într-un singur sens.

Pentru a ne putea referi la celulele benzii BI le vom numerota. La fiecare avans numărul celulei curente crește cu 1. În figura I.4 se prezintă simplificat un caz în care celula curentă este a patra celulă a benzii BI.

5. *Capul de înscriere* CS (asemănător capului de înregistrare a benzii magnetice la magnetofon sau dispozitivului ce imprimă sumele de plată pe banda de hîrtie a mașinii electromecanice de casă aflată în multe din magazine) folosește la depunerea datelor pe banda BE. Considerentele prezentate privind celula curentă și deplasarea celulă cu celulă ale benzii BI sînt valabile și pentru capul de înscriere și banda BE. Convențional, vom considera că celulele benzii BE care nu au trecut încă de capul de înscriere conțin caracterul blanc.

Starea capetelor este dată de numărul ce reprezintă poziția lor pe benzile respective.

6. *Funcționarea* SPDS va începe întotdeauna avînd capetele CC și CS poziționate pe primele celule ale benzilor respective.

În continuare, calculatorul C va citi date de pe BI, le va prelucra (după cum vom vedea mai departe) și, din cînd în cînd, va scrie valori (date) pe BE. În timpul funcționării SPDS, cele două benzi se vor deplasa în sens unic, în mod independent una de alta.

La un moment dat funcționarea SPDS va înceta. Șirul de date citite de pe BI pînă la acel moment formează *datele de intrare* ale prelucrării realizate de SPDS, iar șirul de date înscrise pe BE pînă la acel moment formează *rezultatele* prelucrării.

> Funcționarea SPDS într-o prelucrare are ca rol transformarea datelor de intrare în rezultate.

Funcționarea poate fi văzută și ca o transformare a *fișierului de intrare* în *fișierul de ieșire.*

1.1.2. Funcționarea calculatorului C

Să vedem cum se realizează prelucrările în cadrul calculatorului C. Pentru aceasta vom analiza, mai întîi, structura și funcționarea celorlalte componente ale calculatorului : memoria, dispozitivul de prelucrare și dispozitivul de comandă.

1. Memoria M este alcătuită dintr-o mulțime de celule ce pot găzdui date din cele patru tipuri menționate. Putem să ne formăm imagini intuitive asupra memoriei gîndindu-ne la un fagure de albine, la un dulap cu multe sertare identice sau la casetele pentru bagaje din gări. Fiecare celulă a fagurelui sau fiecare sertar al dulapului ar putea fi privite ca celule ale memoriei M. În asemenea celule pot fi înscrise date sau din ele pot fi extrase date. Dacă vrem să înscriem o dată într-o celulă ce conține deja una, noua dată va lua locul celei vechi, cea veche dispărînd definitiv din celulă. În schimb, extragerea unei date dintr-o celulă a memoriei nu modifică conținutul celulei, operația executîndu-se prin copierea valorii.

> Într-o celulă a memoriei nu poate exista decît cel mult o dată (de tip numeric, logic sau șir de caractere[1]). În celulele memoriei pot fi „înscrise" date și din celule pot fi „citite" date.

O celulă în care nu s-a înscris nici o astfel de dată se numește neinițializată și orice încercare de a o citi este o eroare pe care calculatorul o semnalează.

---

[1] La calculatoarele reale capacitatea unei celule de memorie este limitată, ceea ce afectează uneori serios prelucrările. Calculatorul *C* fiind imaginar, ne putem permite să nu-i limităm capacitatea celulelor memoriei (de exemplu, să memorăm orice număr sau orice șir finit de caractere).

Pentru a putea citi și scrie din/în celulele memoriei este necesar să ne putem referi la ele. În limbajele de programare, deci și în LPS, referirea la celulele memoriei se face printr-un nume simbolic, numele celulei. În cadrul unei prelucrări necesitate de rezolvarea unei probleme folosind SPDS, un număr finit de celule din M capătă fiecare cîte un nume pe care și-l păstrează în tot cursul prelucrării. Ansamblul acestor celule formează *zona datelor* prelucrării. În cele ce urmează vom scrie numele celulelor din zona datelor cu litere mici : *x*, *alfa*, *b*, *max* ș.a.m.d. În reprezentările grafice ale memoriei, cum este cea din figura I.5, celulele din zona datelor vor fi reprezentate compact, fiecărei celule asociindu-i-se conținutul, numit *valoarea* celulei, și numele ei, numit uneori și *adresa* sau *referința* celulei.

Starea memoriei în cursul unei prelucrări este dată de mulțimea perechilor (adresă ; valoare) ale tuturor celulelor zonei de date.

Pentru zona datelor din figura I.5 starea memoriei este dată de mulțimea

{(*x* : —15), (*a* ; **fals**), (*max* ; 153), (*nume* ; 'ION'), (*val* ; 54,7), (*m* ; 10), (log ; **adevărat**), (*cod* ; 'F5'), (*an* ; 1980)}.

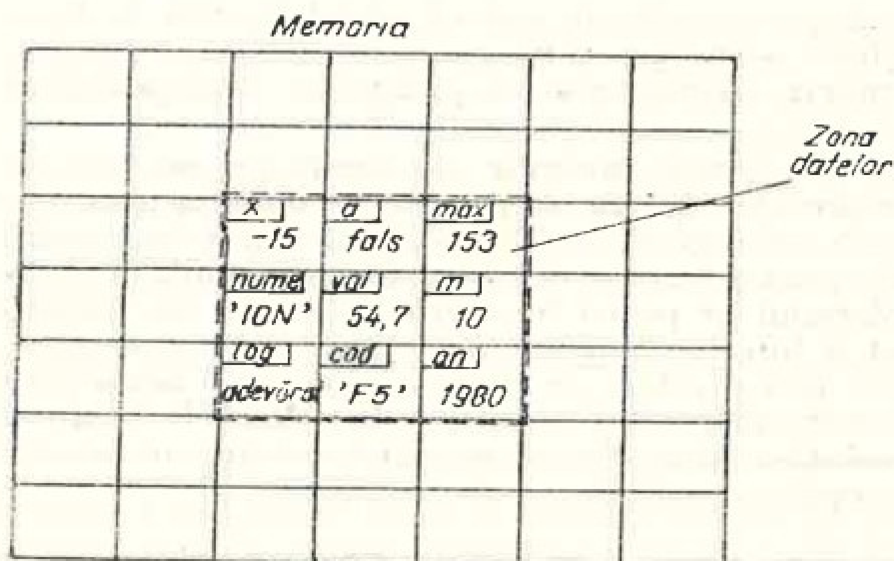


Fig. I.5

2. *Dispozitivul de prelucrare* DP realizează calculul valorii unei funcții pentru care se cunosc valorile argumentelor. DP este reprezentat schematic în figura I.6. Valorile argumentelor reprezintă „intrarea" în DP, iar valoarea funcției „ieșirea" din DP.

În cele ce urmează vom considera că funcțiile realizate de DP pot fi descrise prin expresii ce conțin operații cunoscute, cum ar fi :

a) operații algebrice : adunare, scădere, înmulțire, împărțire și ridicare la putere,

b) operații logice : negație, conjuncție și disjuncție,

c) compararea a două valori numerice în scopul stabilirii unei valori logice (**adevărat** sau **fals**) după cum cele două valori sînt sau nu într-una din relațiile : $<$, $\leqslant$, $=$, $\neq$, $\geqslant$, $>$.

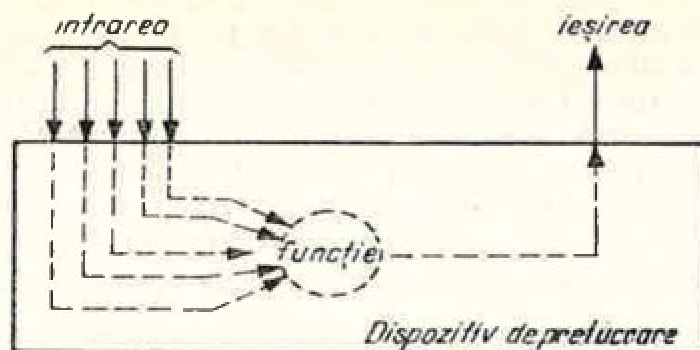


Fig. I.6

În fine, expresiile ce descriu funcţiile realizate de DP pot conţine şi funcţii trigonometrice, exponenţiale, logaritmice etc. (în general, funcţii elementare).

Realizarea calculului valorii unei funcţii de către DP este bine să fie privită ca o secvenţă de operaţii de tipurile a), b), c), precum şi evaluări de funcţii cunoscute deja amintite. Vom exemplifica acest mod de a privi activitatea dispozitivului de prelucrare pe două cazuri prezentate schematic în figurile I.7, *a* şi I.7, *b*.

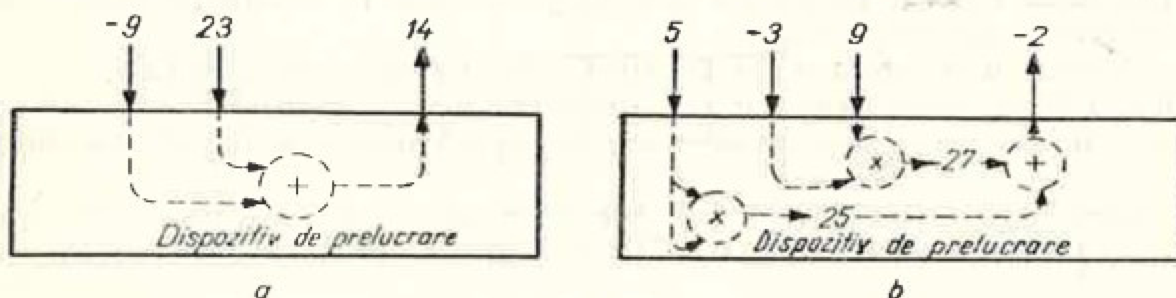


Fig. I.7

În figura I.7, *a* se prezintă evaluarea unei funcţii $f(x, y) = x + y$ pentru intrările $x = -9$ şi $y = 23$. Valoarea funcţiei, 14, rezultă din operaţia de adunare a celor două valori : -9 şi 23.

În figura I.7, *b* se prezintă evaluarea unei funcţii $g(x, y, z) = x \cdot x + y \cdot z$ pentru intrările $x = 5$, $y = -3$, $z = 9$. Se observă că DP realizează o secvenţă de trei operaţii. Ordinea operaţiilor este impusă. Ea rezultă din regulile binecunoscute de evaluare a expresiilor algebrice ; înmulţire, din nou înmulţire şi la urmă adunare. Rezultatele intermediare ale unor operaţii intervin, după cum se vede, în operaţii ulterioare.

În figura I.8 se prezintă evaluarea unei funcţii $h(x, y) = (x + y > 0) \wedge \wedge (y \neq 5)$ a cărei valoare este **fals** pentru cazul $x = -7$, $y = 3$. Din nou

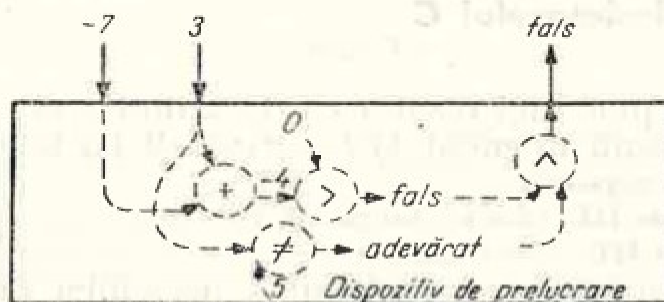


Fig. I.8

regulile de evaluare a expresiilor algebrice pe baza priorităților operațiilor asigură o secvență unică de operații : adunare, comparare la $>$, comparare la $\neq$, conjuncție. În fine, expresia conține două constante : 0 și 5 ale căror valori considerăm că sînt asigurate de către DP, astfel încît doar valorile argumentelor să constituie „intrarea" lui DP.

3. *Dispozitivul de comandă* DC este locul în care se iau decizii asupra modului de funcționare a calculatorului C. Pe baza deciziilor luate, DC comandă executarea unor operații din setul celor patru operații de bază ale calculatorului.

> Operațiile de bază ale calculatorului C sînt : citirea, atribuirea, scrierea și oprirea.

a) Efectuarea unei operații de *citire* a valorii celulei curente de pe BI și depunerea ei într-o celulă dată din M necesită din partea DC comenzi pentru executarea următoarelor acțiuni :

— extragerea de către CC din celula curentă a lui BI a informației conținute de aceasta,

— plasarea valorii în celula din M precizată în operație (deci : *modificarea valorii celulei respective*),

— avansul cu o celulă a lui BI (deci : *modificarea poziției* CC).

b) Efectuarea unei operații de *atribuire* a valorii unei funcții date unei celule din M, de asemenea dată, necesită următoarele acțiuni comandate de DC :

— copierea valorilor din celulele lui M ce reprezintă argumentele funcției,

— evaluarea funcției de către DP folosind valorile argumentelor extrase din M,

— înscrierea valorii rezultatului în celula din M precizată în operație (deci : *modificarea valorii celulei din M*).

c) Efectuarea unei operații de *scriere* a unei valori dintr-o celulă dată din M în celula curentă de pe banda BE necesită următoarele acțiuni comandate de DC :

— copierea din celula dată a lui M a valorii conținute,

— înscrierea valorii în celula curentă a benzii BE (deci : *modificarea conținutului celulei curente a lui BE*),

— avansul cu o celulă a lui BE (deci : *modificarea poziției CS*).

d) Operația de *oprire* are ca efect blocarea funcționării calculatorului C de către DC.

1.1.3. Starea calculatorului C

Din cele arătate pînă aici rezultă că dacă dorim să cunoaștem *starea calculatorului* la un anumit moment al funcționării lui în cadrul unei aplicații, este suficient să cunoaștem :

a) poziția CC pe BI (deci *starea* CC),

b) poziția CS pe BE (deci *starea* CS),

c) zona datelor aplicației ca mulțime a perechilor (adresă, valoare), deci *starea* memoriei.

În figura I.9 se prezintă două stări distincte ale SPDS despărțite de execuția unei operații de citire în celula x a memoriei. Se observă de la figura I.9, a la figura I.9, b creșterea cu 1 a poziției CC (de la 4 la 5) și modificarea conținutului celulei x cu valoarea existentă în celula curentă (a 4-a) citită. În figura I.9, a starea calculatorului C este dată de tripletul :

$$(4,\ 1, \{(i\ ;\ 4),\ (x\ ;\ -2),\ (m\ ;\ 13)\})$$

iar în figura I.9, b de tripletul :

$$(5,\ 1,\ \{(i\ ;\ 4),\ (x\ ;\ 24),\ (m\ ;\ 13)\}).$$

Operațiile de bază : citire, atribuire și scriere modifică starea calculatorului astfel :
a) *citirea* modifică poziția capului de citire și valoarea celulei din M afectată de citire ;
b) *atribuirea* modifică valoarea celulei din M unde se depune rezultatul;
c) *scrierea* modifică poziția capului de înscriere.

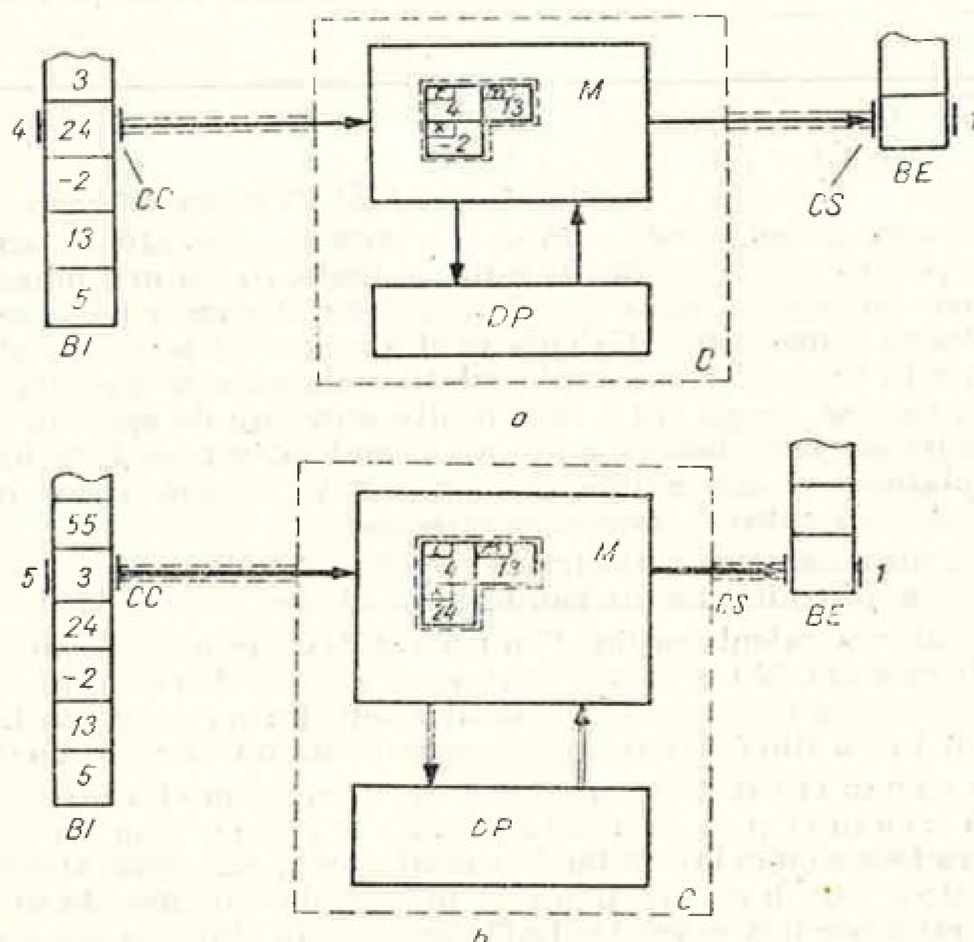


Fig. I.9

Se observă că adăugind la starea lui C conținutul benzilor BI și BE obținem un sistem închis a cărui succesiune a stărilor, numită și *evoluție*, depinde doar de șirul operațiilor prin care obligăm sistemul să treacă. Păstrind mereu un același șir al operațiilor și modificind pentru fiecare nouă evoluție conținutul benzii de intrare BI, obținem la sfirșitul evoluțiilor (atunci cind se execută operația de oprire) benzi de ieșire cu conținuturi diferite (rezultatele). Totalitatea perechilor (date de intrare, rezultate) caracterizează, după cum vom vedea în paragraful următor, șirul operațiilor de executat.

1.1.4. Programarea calculatorului C

1. Am denumit *date de intrare* şirul datelor aflate în celulele citite ale BI şi *rezultate* şirul datelor aflate în celulele scrise ale BE la sfîrşitul prelucrării (după execuţia operaţiei de oprire). Transformarea datelor de intrare în rezultate este consecinţa execuţiei unei secvenţe de operaţii de bază.

De obicei, mulţimea datelor de intrare pentru care putem executa o anumită secvenţă de operaţii este dată prin specificarea problemei. Să notăm cu D această mulţime numită şi *domeniul* secvenţei de operaţii.

Să considerăm şi mulţimea tuturor perechilor (date de intrare, rezultate) pentru datele de intrare din D. Să notăm cu K această mulţime. Ea se numeşte *comportare externă* a secvenţei de operaţii pentru domeniul D. Comportarea externă este a doua componentă a specificării oricărei probleme.

> Dîndu-se o pereche (D, K), *a programa calculatorul C* înseamnă a determina o secvenţă de operaţii de bază care, executate în ordinea secvenţei pentru toate datele din D, să aibă mulţimea K drept comportare externă.

2. Să considerăm mulţimile:

$$D = \{(x, y) \mid x, y \in \mathbf{R}\}$$
$$K = \{((x, y), z) \mid (x, y) \in D, z = \max(x, y)\}.$$

Se poate observa că nu există nici o secvenţă de operaţii de bază care să scrie pe BE, pentru orice pereche de numere reale, pe cel mai mare dintre ele. Ar trebui două secvenţe: una care să scrie primul număr citit, cealaltă care să scrie al doilea număr citit. Calculatorul va executa una sau alta din cele două secvenţe în funcţie de mărimile relative ale celor două numere. Asemenea operaţii, care să aleagă între mai multe secvenţe de operaţii pe una singură în funcţie de îndeplinirea anumitor condiţii, nu au fost încă definite pentru calculatorul C. Ele există însă în cazul calculatoarelor reale şi sînt strîns legate de conceptul de *program memorat*.

În cele ce urmează vom introduce şi pentru calculatorul C, posibilitatea de memorare a programului, mărindu-i astfel autonomia de funcţionare.

Din prezentarea calculatorului C nu a rezultat de unde primeşte comenzile pe care le execută. Nu ne-a interesat acest aspect, deoarece ne-am îndreptat atenţia spre ceea ce poate face calculatorul din punct de vedere al prelucrării, acest lucru fiind descris prin secvenţe de operaţii de bază.

Pentru a comanda calculatorul C, cel mai simplu mod ar fi să-i dăm comandă după comandă prin intermediul unor butoane, comutatoare etc. În felul acesta au fost comandate primele calculatoare, iar astăzi sînt comandate unele calculatoare de buzunar şi unele maşini de calculat de birou. Într-o astfel de situaţie omul urmăreşte desfăşurarea calculului pas cu pas şi, în funcţie de rezultatele intermediare, ia decizii asupra modului în care continuă prelucrarea, deci asupra comenzilor următoare. Se ştie că această modalitate este dezavantajoasă din punct de vedere al vitezei de calcul deoarece nu se exploatează viteza cu care calculează calculatorul, mult mai mare în raport cu viteza de reacţie a omului.

Înzestrarea calculatorului cu posibilitatea de a decide asupra următoarei operaţii de executat necesită „cunoaşterea" de către calculator a tuturor posibilităţilor de continuare a calculului, deci cunoaşterea de la început a tu-

turor calculelor ce trebuie să le execute calculatorul în diversele situații posibile, cu alte cuvinte a întregului program. Ajungem astfel la un concept extrem de important: *programul memorat*. Calculatoarele moderne funcționează aproape în exclusivitate „comandate" de un program memorat.

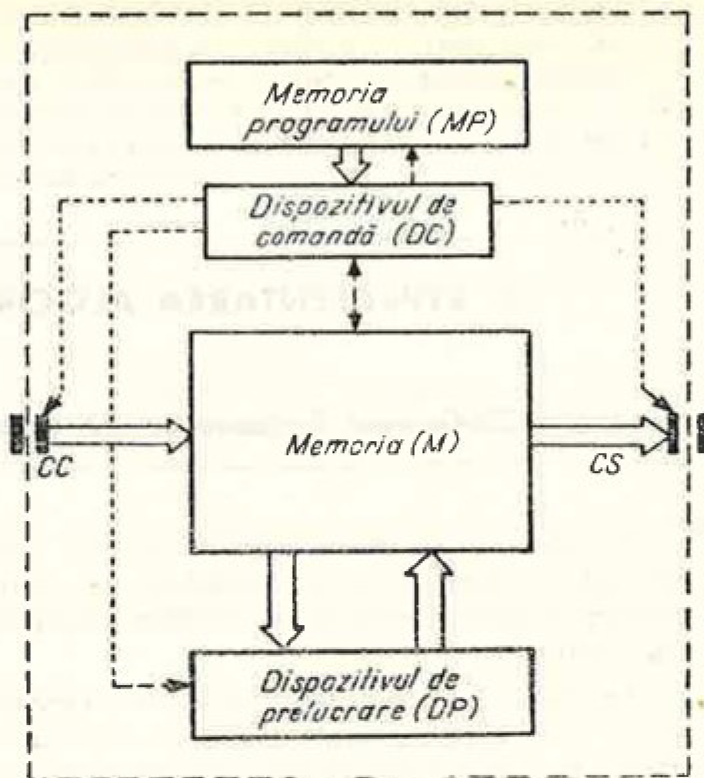


Fig. I.10

Programul este memorat într-o „zonă a programului" aflată în memoria calculatorului, zonă distinctă în cazurile reale doar din punct de vedere al modului de interpretare a informațiilor din ea. Dispozitivul de comandă își extrage din zona respectivă instrucțiunile una după cealaltă, conform unor reguli bine stabilite.

În cazul calculatorului C putem completa schema din figura I.3 cu o *memorie a programului MP* în care utilizatorul depune, printr-un procedeu oarecare de încărcare, instrucțiunile programului (fig. I.10). Nu ne interesează cum este organizată memoria MP, cum sînt reprezentate instrucțiunile și nici cum dispozitivul DC își extrage de aici următoarea instrucțiune de executat.

Ceea ce este important pentru noi este cunoașterea acelor operații ale calculatorului C care permit schimbarea locului din MP de unde DC își extrage următoarea instrucțiune de executat. Aceste operații sînt altele decît cele de bază. Ele se referă la dirijarea în memoria MP a indicatorului următoarei instrucțiuni de executat. Ele se numesc *operații de control*. Rolul operațiilor de control va fi prezentat în paragraful I.3 cînd se vor descrie instrucțiunile condiționate și de ciclare.

ÎNTREBĂRI

1. Care sînt componentele SPDS? Cum sînt alcătuite benzile de intrare și ieșire?
2. Cum funcționează SPDS?
3. Care sînt componentele calculatorului C și care sînt relațiile între ele?
4. Cum funcționează calculatorul C?
5. Care este structura memoriei M? Ce conțin celulele memoriei și cum are loc accesul la ele?
6. Cum funcționează dispozitivul de prelucrare DP? Care sînt operațiile realizate de DP?
7. În relația memorie-dispozitiv de prelucrare, memoria este elementul pasiv, cel care suportă operațiile, iar dispozitivul de prelucrare este elementul activ, cel care realizează prelucrările aritmetico-logice. Cum se realizează această relație?
8. Cum își manifestă dispozitivul de comandă DC funcția de conducere a activității calculatorului C?

9. Care sînt operaţiile de bază ale calculatorului C ?

10. Cum poate fi comandată din exterior o maşină de calcul ?

11*. Care sînt avantajele programului memorat, ce implicaţii are acest concept asupra autonomiei de lucru a calculatoarelor ?

12*. Care sînt informaţiile de stare ale SPDS şi ale calculatorului C ? Descrieţi operaţiile de bază în termenii modificării informaţiilor de stare.

I.2. REPREZENTAREA ALGORITMILOR ÎN LIMBAJUL LPS

I.2.1. Definirea limbajelor de programare

Limbajele de programare se pot învăţa cum se învaţă şi alte limbaje artificiale[1] : limbajul formulelor şi expresiilor matematice, limbajul logicii, limbajul formulelor şi ecuaţiilor chimice, limbajul schemelor circuitelor electrice etc.

În mod special, în cazul limbajelor de programare, definirea lor trebuie să fie precisă. Prin analogie cu limbajul natural[2] un program este o „propoziţie" într-un limbaj de programare.

El descrie o anumită funcţionare a calculatorului. Care este această funcţionare rezultă din interpretarea programului respectiv conform definiţiei limbajului. Definiţia trebuie deci să fie clară pentru ca *orice* program să fie înţeles la fel de *orice* utilizator şi anume exact în sensul pe care calculatorul îl atribuie programului respectiv.

Şi în cazul limbajelor de programare, ca şi în cazul limbajelor naturale, definirea se face pe părţi componente. Se ştie că în gramatica unui limbaj natural găsim reguli de forma : o frază este alcătuită din una sau mai multe propoziţii, o propoziţie este formată din subiect, predicat şi eventual alte părţi de propoziţie etc.

Vom regăsi deci şi la limbajele de programare asemenea „părţi de propoziţie". Despre una din ele deja am amintit : instrucţiunea. Se ştie că un program este format din instrucţiuni.

Vom numi *elemente componente ale limbajului* acele construcţii din limbaj caracterizate de anumite particularităţi comune de scriere şi semnificaţie.

De exemplu, în limba română noţiunea de predicat se caracterizează atît la nivelul structurii (cum este alcătuit) : un verb sau un verb cu un nume predicativ, cît şi la nivelul semnificaţiei : exprimă o acţiune etc. Într-un lim-

---

— Exerciţiile cu un grad sporit de dificultate au fost notate cu *. Dificultatea poate proveni din mai multe cauze :
— răspunsul antrenează cunoştinţe din clasele precedente,
— răspunsul presupune o discuţie asupra soluţiilor posibile,
— răspunsul necesită aprofundarea unor cunoştinţe de bază.

[1] Limbaj artificial — Limbaj creat deliberat de un grup restrîns de oameni pentru comunicare într-un anumit domeniu al ştiinţei sau tehnicii.

[2] Limbaj natural — Limbaj apărut spontan, de-a lungul istoriei în mari colectivităţi umane în cadrul procesului de muncă în vederea comunicării între membrii colectivităţilor respective.

baj de programare, o instrucţiune are reguli precise de scriere, iar semnificaţia ei constă dintr-un anumit efect pe care îl are execuţia instrucţiunii de către un calculator asupra stării acestuia.

Cele două niveluri : cel al scrierii şi cel al semnificaţiei, apar foarte bine conturate în cazul limbajelor de programare.

> Orice element component al unui limbaj de programare trebuie definit la două niveluri :
> 1° cum se formează (scrie) corect *(nivelul sintactic)*,
> 2° ce semnificaţie are pentru utilizator, semnificaţie înţeleasă ca efect produs asupra funcţionării unui calculator *(nivelul semantic)*.

Regulile de formare a programelor corecte alcătuiesc *sintaxa* limbajului de programare.

Regulile ce determină semnificaţia programelor corecte alcătuiesc *semantica* limbajului de programare.

În cele ce urmează vom prezenta un limbaj folosit pentru descrierea algoritmilor : limbajul LPS.

Limbajul LPS va fi folosit ca o fază intermediară în scrierea programelor, alternativă posibilă, de cele mai multe ori preferabilă, schemelor logice.

1.2.2. Instrucţiunile de bază ale limbajului LPS

1. Vom introduce întîi cele mai simple instrucţiuni ale LPS. Ele corespund ca semnificaţie operaţiilor de bază ale calculatorului C : citirea, scrierea, atribuirea şi oprirea. Pentru a da o reprezentare, o descriere simbolică acestor operaţii, să ne imaginăm că putem comanda calculatorul C prin comenzi scrise : citeşte !, scrie ! etc. Sub această formă comenzile apar însă incomplete. În cazul comenzii „citeşte!", de exemplu, calculatorul C ştie de unde să citească (de pe banda BI) şi ce să citească (valoarea din celula curentă), dar nu va şti ce să facă cu valoarea citită, nu va şti în care celulă din M să o plaseze. De aceea ordinul trebuie completat cu numele celulei în care se depune valoarea citită

citeşte *x*

Pentru a marca mai bine partea fixă a comenzii, cuvîntul „citeşte", o vom scrie îngroşat (cu litere aldine) astfel :

citeşte *x*

În fine, deoarece în locul lui *x* poate fi *y* sau *alfa* sau *max* sau orice alt nume de celulă din memorie, vom defini scrierea corectă a comenzii sub forma :

> **citeşte** < *variabilă* >

De reținut următoarele convenții folosite în această definiție:

Cuvîntul *<variabilă>*[1] scris cu litere cursive între parantezele unghiulare < și > desemnează elemente ale unei mulțimi (mulțimea numelor celulelor memoriei M). În acest fel, *<variabilă>* din punct de vedere sintactic este „un cuvînt cu litere mici", iar semantic: „o celulă din memoria calculatorului".

Vom numi *instrucțiune* o reprezentare simbolică a unei comenzi date calculatorului. În cazul de față instrucțiunea descrisă se numește *instrucțiune de citire*. Dacă *a*, *beta*, *alfa* sînt variabile, atunci:

citește *a*

citește *beta*

citește *alfa*

sînt instrucțiuni de citire.

În figura I.11 se prezintă efectul instrucțiunii **citește** *a* asupra calculatorului C în cazul în care celula curentă a benzii de intrare conține numărul 15. Se observă că efectul este cel al operației de citire a calculatorului C.

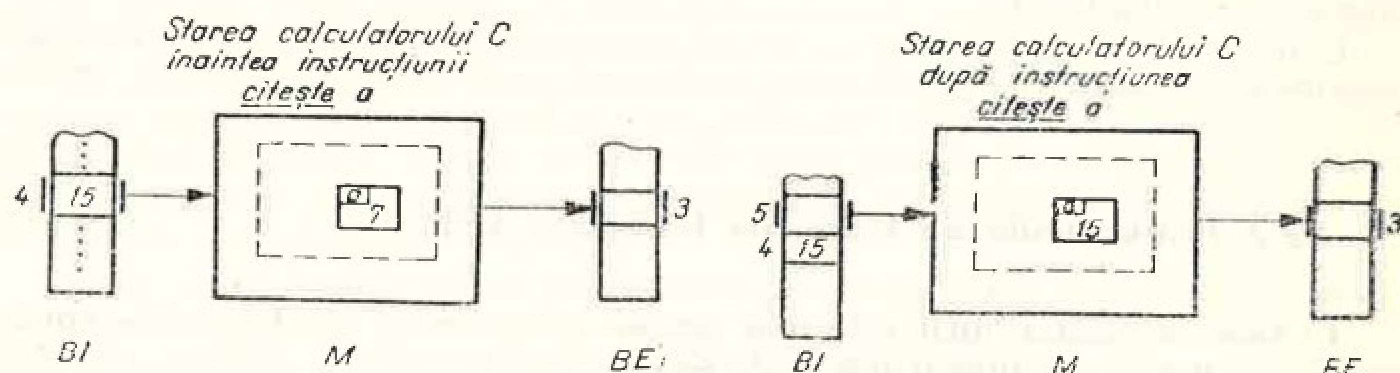


Fig. I.11

2. Asemănător, instrucțiunea de scriere din LPS, ca „ordin scris" adresat calculatorului C, o vom nota astfel:

scrie *<variabilă>*

Deci **scrie** *p* sau **scrie** *limită* vor fi instrucțiuni de scriere corecte în LPS. Efectul unei instrucțiuni de scriere asupra calculatorului C este cel al operației de scriere. De exemplu, în figura I.12 se prezintă efectul instrucțiunii **scrie** *p* cînd în celula *p* din memoria calculatorului se află valoarea 13.

---

[1] Denumirea de variabilă provine din limbajul expresiilor algebrice. O variabilă este un nume (literă sau cuvînt, cu indici sau fără) ce desemnează *orice* valoare dintr-o mulțime. Mulțimea formează domeniul variabilei respective. De exemplu, în expresia $x^5 + 1$, $x \in \mathbf{R}$, x este o variabilă al cărei domeniu este mulțimea numerelor reale. În cazul limbajelor de programare, variabilele desemnează întotdeauna o pereche *celulă-valoare memorată*, celula rămînînd mereu aceeași în cursul execuției unui program, în timp ce valoarea memorată se poate schimba, dar rămînînd într-o aceeași mulțime dată de tipul variabilei.

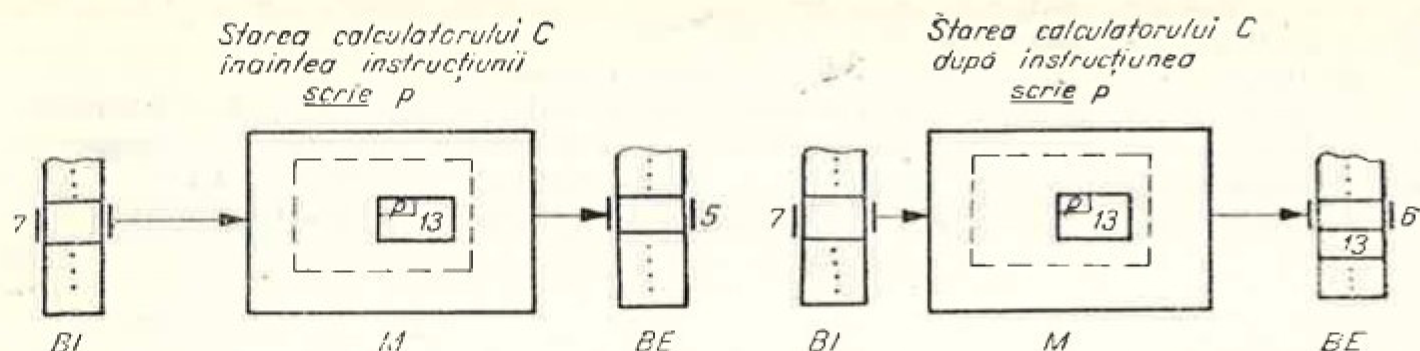


Fig. I.12

3. Pentru a efectua o operație de atribuire vom comanda calculatorul C cu :

atribuie *<variabilă>* ← *<expresie>*

unde *<variabilă>* reprezintă numele celulei din memorie în care se depune rezultatul calculului descris prin *<expresie>*.

De exemplu

atribuie $a \leftarrow 5$
atribuie $vit \leftarrow sp/timp$
atribuie $x \leftarrow (-b + \sqrt{delta})/(2*a)$
atribuie $y \leftarrow (exp(x) \quad - exp(-x))/2$
atribuie $alfa \leftarrow \arcsin(\sin^2(x) - \cos^2(x)) - 1$
sînt instrucțiuni de atribuire. Să observăm :

a) **atribuie** și ← sînt părțile fixe ale instrucțiunii, *<variabilă>* și *<expresie>* au reprezentări diverse, în funcție de necesități.

b) sintaxa pentru *<expresie>* este cea a expresiilor algebrice în care apar variabile și constante, cu operații de adunare, scădere, înmulțire și împărțire, cu evaluări de funcții cunoscute etc.

Să remarcăm în plus semnul $*$ pentru înmulțire, / pentru împărțire, notația *exp* pentru funcția exponențială și utilizarea exclusivă a parantezelor (,) în expresii.

În figura I.13 se prezintă modificările din memoria calculatorului C la primirea „ordinului" : **atribuie** $delta \leftarrow b^2 - 4*a*c$

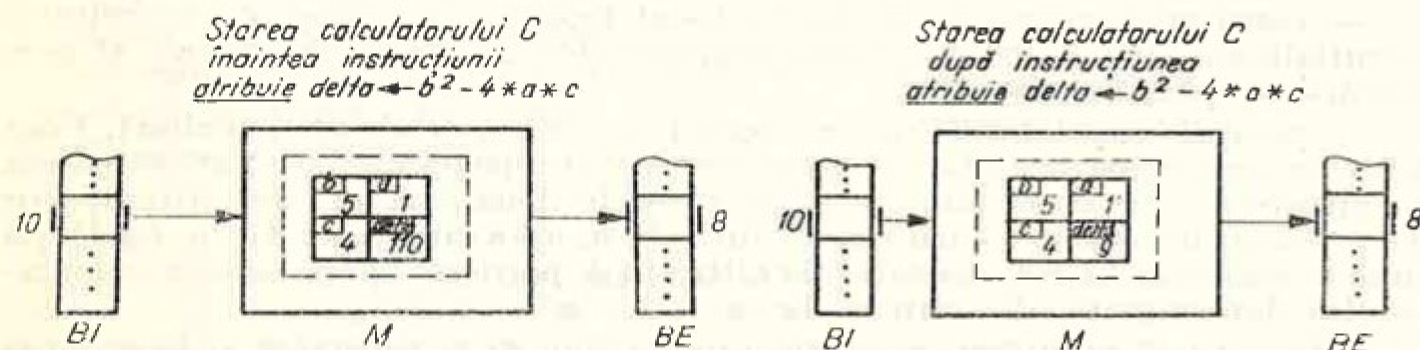


Fig. I.13

Putem să folosim instrucțiunea de atribuire și pentru valori logice. De exemplu: **atribuie** $l \leftarrow$ **adevărat**

atribuie $z \leftarrow n \lor w$ (dacă n și w conțin valori logice).

În general, expresiile folosite pot descrie calcule atît cu valori numerice cît și logice. De exemplu, dacă celulele a și b conțin valorile 5 și respectiv 10, evaluarea expresiei $a^2 \leqslant b \lor a > b^2$ va produce ca rezultat **fals**.

4. Operația de oprire a calculatorului C o vom scrie în limbajul LPS:

stop

I.2.3. Secvența de instrucțiuni

1. După ce am văzut cum se scriu instrucțiunile de bază ale limbajului LPS, să vedem cum descriem în acest limbaj înlănțuirea operațiilor de bază ale calculatorului C, adică execuția lor succesivă. În marea majoritate a limbajelor de programare ordinea normală de execuție a operațiilor este dată chiar de ordinea în care se scriu instrucțiunile.

Să luăm ca exemplu citirea a două valori succesive de pe BI și plasarea lor în memoria M în celulele a, respectiv b. În figura I.14 sînt prezentate trei stări ale calculatorului C corespunzătoare celor 3 momente importante: înaintea execuției celor două citiri, între citiri și după execuția citirilor. În această figură folosim și o nouă reprezentare a stării sistemului SPDS. Ea

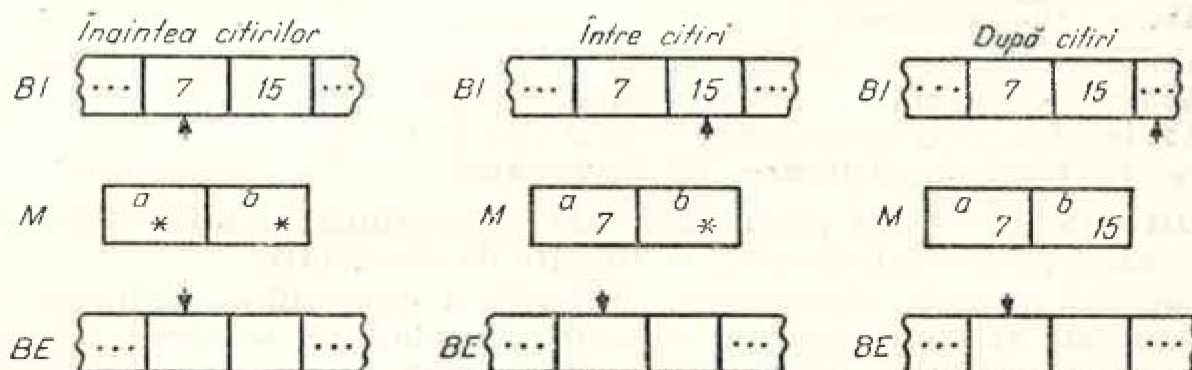


Fig. I.14

este mai simplă decît cea folosită în figurile I.11—I.13. Noua reprezentare cuprinde toate informațiile necesare:

— conținutul benzii BI (în partea superioară a figurii),

— conținutul benzii BE (în partea inferioară a figurii),

— conținutul memoriei M (la mijlocul figurii). Convenim ca în celulele neinițializate sau care conțin valori neinteresante din punct de vedere al prelucrării să plasăm semnul *,

— pozițiile capetelor CC și CS (săgețile ce indică celule ale benzilor). Vom folosi în continuare această reprezentare. În cazul particular studiat, pe banda de intrare se află două numere: 7 și 15. Cele două comenzi de citire se vor da evident în ordine: întîi citirea lui 7 în a, apoi citirea lui 15 în b. După cum se vede din figură, operația de citire în b pornește de la starea calculatorului determinată de citirea în a.

Este natural ca ordinea executării operațiilor să o respectăm și în scrierea instrucțiunilor în LPS: întîi **citește** a, apoi **citește** b.

Pentru a păstra claritatea programului LPS vom scrie cele două instrucțiuni una sub alta :

citește a

citește b

Un asemenea grup de instrucțiuni scrise una după cealaltă îl numim *secvență de instrucțiuni*. Într-o secvență de instrucțiuni pot fi oricîte instrucțiuni (în particular chiar și una singură). De exemplu :

citește a

scrie a

este o secvență de instrucțiuni. Dacă dorim să știm ce prelucrări determină această secvență va trebui să „înlănțuim" efectele celor două instrucțiuni în ordinea scrierii lor : după ce s-a citit în celula a valoarea curentă de pe banda de intrare, se scrie pe banda de ieșire valoarea din celula a care este *exact* cea citită puțin mai înainte.

2. În cazul în care într-o secvență apar instrucțiuni de bază de același tip, consecutive, vom introduce o notație simplificată scriind comanda **citește**, **scrie** sau **atribuie** o singură dată după cum vom înlănțui părțile lor variabile exact în ordinea în care au apărut în secvență. Deci :

citește a

citește b

le putem scrie :

citește a, b

(nu **citește** b, a care este echivalent cu : **citește** b
citește a)

De asemenea :

scrie *alfa*

scrie *beta*

scrie *omega*

le putem scrie :

scrie *alfa*, *beta*, *omega*.

În fine, în cazul atribuirilor vom folosi semnul & („și" comercial) pentru a lega atribuirile succesive. De exemplu, secvența :

atribuie $b \leftarrow a + c$
atribuie $d \leftarrow a - c$
atribuie $e \leftarrow b^2 - d^2$

o putem rescrie compact astfel :

atribuie $b \leftarrow a + c$ & $d \leftarrow a - c$ & $e \leftarrow b^2 - d^2$

sau dacă expresiile sînt lungi :

[**atribuie** $b \leftarrow a + c$
& $d \leftarrow a - c$
■ & $e \leftarrow b^2 - d^2$

S-a marcat cu ■ plasat sub **atribuie** sfîrșitul secvenței celor trei atribuiri. Arcul folosit între **atribuie** și ■ evidențiază o dată în plus unitatea celor trei instrucțiuni, faptul că ele formează o secvență de instrucțiuni de același tip pe care le-am scris pe mai multe rînduri.

Limbajul LPS nefiind un limbaj de programare propriu-zis, ci un limbaj de comunicare a algoritmilor, ne putem permite abateri în anumite limite de la regulile de scriere prezentate. Abaterile nu trebuie însă să afecteze semantica instrucţiunilor şi nici să micşoreze claritatea descrierii algoritmului. După cum se va constata în exemplele prezentate în continuare, ne vom permite libertăţi în scrierea unor elemente ale reprezentării algoritmilor în LPS cum ar fi expresiile, sau în compactarea secvenţelor de instrucţiuni ; niciodată însă nu vom afecta cu aceste abateri structura programului rezultată din incluziunea instrucţiunilor unele în altele, incluziune ce va apărea evidentă în cazul instrucţiunilor de control.

Algoritmul I

Ne propunem să descriem în limbajul LPS o prelucrare de date simplă realizată de calculatorul C.

Fie o bandă de intrare pe care se află două numere întregi. Să se descrie prelucrarea realizată de calculatorul C pentru ca cele două numere întregi să fie scrise în aceeaşi ordine pe banda de ieşire.

Algoritmul dorit trebuie să descrie în termenii celor 4 instrucţiuni de bază înlănţuirea operaţiilor de bază ale calculatorului C care asigură realizarea prelucrării cerute.

Pentru aceasta, calculatorul va citi mai întîi numerele, memorîndu-le, şi apoi le va înscrie pe BE. În LPS acest lucru se scrie astfel :

citeşte *a*

citeşte *b*

scrie *a*

scrie *b*

Figura I.15 prezintă cele 5 stări ale calculatorului C în cazul în care BI conţine numerele 5 şi —17.

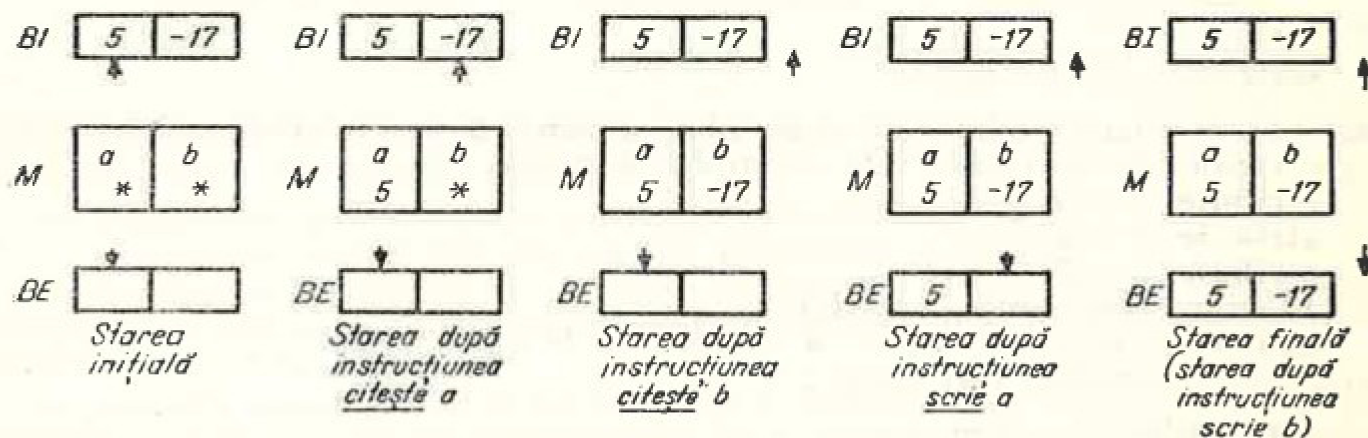


Fig. I.15

Observaţii. Zona datelor pentru prelucrarea descrisă mai sus cuprinde două celule: *a* şi *b*. Numele lor sînt tocmai variabilele folosite în secvenţă. Se observă că este neimportantă alegerea numelor variabilelor. În general, este bine să le alegem cît mai sugestive pentru aplicaţia dată.

Să considerăm secvenţa :

citeşte *a*
scrie *a*
citeşte *a*
scrie *a*

zona datelor în prelucrarea descrisă de această secvenţă are o singură celulă, iar prelucrarea este aceeaşi cu cea descrisă de secvenţa de mai sus. Putem compara cele două prelucrări? Putem afirma despre una din ele că este mai bună decît cealaltă? Să observăm că a doua secvenţă descrie o prelucrare ce necesită mai puţine celule de memorie decît prima. Deci, din punct de vedere al memoriei „folosite", această prelucrare este mai bună decît prima. De multe ori însă, în căutarea unor prelucrări mai eficiente din punct de vedere al memoriei folosite, se poate pierde din claritatea descrierii.

1.2.4. Reprezentarea algoritmilor în LPS

1. În paragraful precedent am descris în LPS o prelucrare simplă folosind o secvenţă de patru instrucţiuni de bază. Pentru a fi o reprezentare completă a unui algoritm în LPS, cele patru instrucţiuni trebuie în primul rînd completate cu o instrucţiune care să determine oprirea calculului de îndată ce rezultatele au fost obţinute. Secvenţa, completată cu instrucţiunea de oprire, devine :

citeşte *a*
citeşte *b*
scrie *a*
scrie *b*
stop

În al doilea rînd, reprezentarea completă necesită şi o descriere a tuturor numerelor simbolice folosite în algoritm. Pentru a aprecia utilitatea unei asemenea descrieri să considerăm următoarea secvenţă :

atribuie $a \leftarrow 5$
atribuie $b \leftarrow 3,1$
atribuie $c \leftarrow a * b$
atribuie $d \leftarrow a \vee c$

Din primele două instrucţiuni rezultă că în celula *a* se memorează un număr întreg şi în celula *b* un număr real. Trecînd la a treia instrucţiune, să vedem ce valoare se calculează în vederea memorării ei în *c*. Este o valoare întreagă sau reală ? Întrebarea nu este lipsită de sens deoarece valoarea rezultă din înmulţirea unei valori întregi cu una reală. Dacă în algebră este de la sine înţeles ce se întîmplă cu asemenea expresii mixte, în cazul calculatoarelor trebuie să fim atenţi deoarece, după cum se ştie, numerele reale şi cele întregi au reprezentări interne diferite şi, în consecinţă, operaţii de prelucrare diferite. O operaţie mixtă (cu operanzi de tipuri diferite) cum este cea de mai sus se va executa în două etape : întîi are loc o conversie de la reprezentarea valorii întregi a lui *a* la reprezentarea valorii reale echivalente, după care această valoare se înmulţeşte cu valoarea reală a lui *b*, rezultatul fiind o valoare reală.

În plus, într-un program acţiunea unei operaţii depinde şi de felul operanzilor. Aceeaşi operaţie, să zicem + sau /, se realizează în **Z** dacă operanzii sînt întregi şi în **R** dacă operanzii sînt reali, ceea ce conduce la rezultate diferite. Este deci important să cunoaştem ce fel de valori au operanzii fiecărei operaţii.

Cu a patra instrucţiune lucrurile se complică mai mult deoarece operaţia „sau logic" dintre *a* şi *c* nu are nici un sens, valorile acestor variabile fiind numerice. În asemenea cazuri nu se poate pune problema conversiei implicite ca mai sus, ci pur şi simplu trebuie să o considerăm ca o eroare de programare.

„Detectarea" unor asemenea situaţii se poate face încă înainte de a executa programul dacă s-ar cunoaşte pentru fiecare nume simbolic folosit mulţimea valorilor pe care acesta le poate reprezenta, deci domeniul lui de definiţie. În cazul calculatorului C vom considera că o variabilă se caracterizează, pe lîngă nume şi valoare conţinută şi prin domeniul de definiţie, iar acest domeniu rămîne neschimbat pentru orice execuţie a algoritmului. Pentru exemplul de mai sus, dacă *a* şi *c* au valori numerice, doar dintr-o analiză a textului algoritmului, rezultă că expresia $a \vee c$ nu are sens fără a fi necesară execuţia lui. În plus, specificarea domeniului de definiţie sporeşte claritatea algoritmului, inteligibilitatea lui.

2. În toate limbajele de programare stabilirea domeniului de definiţie al variabilelor se face prin intermediul *declaraţiilor*. Conceptul de declaraţie este de maximă importanţă pentru cei care învaţă să programeze calculatoarele.

Declaraţiile au ca efect asocierea la numele simbolice folosite în program a unor proprietăţi numite *atribute*.

Numele simbolic folosit într-un program pentru a desemna o variabilă se numeşte *identificator*. În majoritatea limbajelor identificatorul este un şir de litere sau cifre în care primul caracter este literă. În cazul LPS am preferat excluderea cifrelor din scrierea identificărilor.

Domeniul de definiţie a unei variabile desemnată printr-un identificator este astfel o mulţime de valori care rezultă din caracterizarea identificatorului prin atribute în cadrul declaraţiilor. Să prezentăm două atribute care sînt des folosite în limbajele de programare :

— *Tipul*. Majoritatea limbajelor au un număr de tipuri primitive : tipul întreg, tipul real, tipul logic, tipul caracter. Ele caracterizează mulţimi de valori cunoscute : **Z, R**, {**adevărat, fals**} şi respectiv un alfabet de caractere. De exemplu :

real *x*

asociază identificatorului *x* tipul real, ceea ce înseamnă că *x* poate lua orice valoare din **R** şi *numai* din **R**.

Asociat fiecărui tip există un set de operaţii de prelucrare a valorilor tipului respectiv. De exemplu, fiecărui tip numeric i se asociază cele patru operaţii : adunare, scădere, înmulţire şi împărţire, rezultatul conservînd tipul operanzilor. Este evident că adunarea valorilor întregi va fi distinctă de adunarea valorilor reale datorită atît domeniilor de definiţie distincte cît şi reprezentărilor în calculator distincte. Tipului logic i se asociază de obicei cele trei operaţii logice etc.

În unele limbaje de programare apărute în ultimii ani sînt înglobate facilităţi ce permit programatorului să-şi creeze tipuri noi de date pe baza tipurilor primitive. Definirea unui nou tip de date într-un limbaj de programare necesită şi definirea operaţiilor tipului respectiv.

— *Structura*. Atributul de structură caracterizează numărul şi organizarea elementelor componente ale unei variabile. Pînă acum toate variabilele întîlnite aveau un singur element. O variabilă cu o asemenea „structură" se mai numeşte *variabilă simplă*. În afara ei, cele mai cunoscute struc-

turi ale variabilelor sînt *tablourile și înregistrările*. În primul caz, o valoare a unei variabile-tablou este formată din valorile de același tip ale componentelor ordonate după unul sau mai multe criterii asemănător vectorilor sau matricelor. Se mai numesc structuri *omogene*. În al doilea caz, o valoare a unei variabile-înregistrare este formată din mai multe valori, de regulă diferite ca tip. Acestea sînt structuri *neomogene*.

Orice tip de structură trebuie să aibă asociat și un mecanism prin care se ajunge la componentele structurii. În cazul tablourilor, mecanismul este *indexarea*. Să presupunem o structură de tip tablou în care elementele componente (toate de același tip) sînt ordonate după un anumit criteriu (asemănător componentelor unui vector). Deci vom putea vorbi de primul element, de al doilea etc. Dacă v este numele variabilei atunci pentru a ne referi la al 5-lea element vom folosi expresia $v(5)$. Valoarea dintre paranteze se numește *indice* și servește deci la localizarea elementului căutat în cadrul structurii. Întreaga expresie desemnează o *variabilă indexată*.

Dacă ordonarea elementelor structurii se face după mai multe criterii vom avea atîția indici cîte criterii sînt. În cazul unei structuri de tip matrice vom avea doi indici : indicele de linii și indicele de coloană. Să presupunem o structură de acest fel numită *mat*. Expresia $mat\ (i + 1, 2)$, dacă valoarea lui i este 6, desemnează elementul aflat pe linia 7 și coloana 3. Din acest exemplu se observă că valorile indicilor pot rezulta și din expresii mai complicate.

În general, un tablou trebuie caracterizat prin numărul de criterii de ordonare, numite *dimensiuni*, iar pentru fiecare dimensiune de valorile pe care le pot lua indicii precum și prin tipul elementelor componente. De exemplu :

mat	tablou, 2 dim. (15, 20)	val. reale

este descrierea în LPS a unui tablou de valori reale cu numele *mat*, cu două dimensiuni pentru care indicii iau valori de la 1 la 15, și, respectiv, de la 1 la 20.

Vom relua și exemplifica cele de mai sus în I.3.3.

În cazul înregistrărilor, în mecanismul de acces se folosesc *selectori* care sînt nume simbolice asociate componentelor. Să presupunem o înregistrare declarată cu numele *data* avînd trei componente (cîmpuri) de tip întreg denumite *zi*, *lună* și *an* :

data = **înregistrare** (**întreg** *zi*, **întreg** *lună*, **întreg** *an*)

și o declarație care asociază numelui x structura neomogenă *data* :

data x

Dacă dorim să avem acces la o componentă vom folosi selectorii *zi*, *lună*, *an* astfel :

x·zi, *x·lună*, *x·an*

Iată cîteva reguli ce se referă la folosirea declarațiilor în limbajele de programare.

1° Orice nume simbolic (identificator) folosit într-un program trebuie declarat[1] în primul rînd ca tip și structură.
2° Orice utilizare a unui identificator nu trebuie să contrazică declarația.
3° Un identificator nu poate fi redeclarat într-o zonă a programului în care este valabilă o primă declarație[2].

3. Pentru limbajul LPS declararea numelor simbolice o vom face printr-un tabel ce însoțește instrucțiunile fiecărui program. Acest tabel colectează toate numele simbolice folosite în program și le caracterizează din punct de vedere al domeniului din care acestea își iau valorile, prin atributele de structură și tip. Pentru aceasta tabelul va avea trei coloane și anume : pentru nume, structură și tip. Dacă este cazul, putem adăuga și alte amănunte despre domeniul valorilor variabilelor, ele fiind utile proiectării și analizei algoritmului.

În figura I.16 se dă reprezentarea completă a algoritmului 1 în LPS folosind și scrierea compactă a instrucțiunilor de bază de același tip, consecutive. S-a folosit prescurtarea v.s. pentru variabilă simplă.

a	v.s.	val. întregi
b	v.s.	val. întregi

citește *a*, *b*
scrie *a*, *b*
stop

Fig. I.16

I.2.5. Probleme

1. Se dau două numere întregi pe banda de intrare în SPDS.
a) Să se scrie algoritmul în LPS care înscrie pe banda de ieșire cele două numere în ordine inversă.
b) Să se prezinte stările succesive ale SPDS în cursul execuției algoritmului pentru o pereche oarecare de valori.

2. Să se scrie algoritmul care înscrie pe banda de ieșire valorile funcțiilor pentru valori date ale argumentelor :
a) $f(x) = e^{-x} \sin x$
b) $g(x, y) = \sqrt[3]{x^3 + y^3}$
c) $\varphi(x, y) = x^5 \cos y^2$
$\psi(x, y) = y^3 + \ln (x^2 + 1)$.

3. Se dă pe banda de intrare o valoare reală reprezentînd valoarea unui unghi dată în radiani. Să se scrie algoritmul în LPS care înscrie pe banda de ieșire valoarea unghiului în grade sexagesimale, minute și secunde. Se consideră $\pi = 3,14159$.

---

[1] Declarația este de obicei explicită : în multe limbaje se folosesc și declarațiile implicite sau prin lipsă. În asemenea cazuri lipsa unei declarații pentru un identificator asociază automat identificatorului un atribut cunoscut, mereu același.

[2] O asemenea restricție trebuie mai bine precizată pentru limbajele de programare cu structură de bloc. Datorită lipsei unor noțiuni necesare precizării ne mulțumim cu această formulare adecvată pentru limbaje cu FORTRAN și COBOL.

I.3. INSTRUCȚIUNILE DE CONTROL ALE LIMBAJULUI LPS

I.3.1. Instrucțiunea condițională

1. Prelucrările descrise de secvențele de instrucțiuni de bază sînt insuficiente. Să luăm un exemplu simplu :

Se dau două numere întregi pe banda de intrare a sistemului SPDS. Să se determine (prin înscriere pe banda de ieșire) cea mai mare valoare din cele două.

Bineînțeles, prima operație va fi să citim numerele și să le plasăm în două celule din memorie, să zicem x și y.

Cum va raționa un om care transmite instrucțiuni de bază calculatorului C pentru a putea rezolva problema ? Evident că întîi trebuie să determine din compararea celor două valori din memorie, care este cea mai mare valoare și de abia apoi să dea comanda de scriere. În cuvinte, raționamentul este următorul :

„dacă valoarea din x este mai mare decît valoarea din y atunci **scrie** x, altfel (adică dacă valoarea din x este mai mică sau egală cu valoarea lui y) **scrie** y".

O formă generală pentru un asemenea raționament foarte frecvent întîlnit de noi în demonstrații sau calcule este :

„dacă *condiție* atunci
execută *ceva*
altfel
execută *altceva*"

Uneori întîlnim și o formă mai simplă :

„dacă *condiție* atunci
execută *ceva*"

în cazul că a doua alternativă nu execută nimic.

Frecvența utilizării unui asemenea raționament ne obligă să introducem o operație de calcul care să-l realizeze, cu atît mai mult cu cît raționamentul nu poate fi descris în termenii celor 4 instrucțiuni de bază. În termenii schemelor logice cele două forme ale raționamentului de mai sus se realizează cu schemele din figura I.17, *a*, *b*.

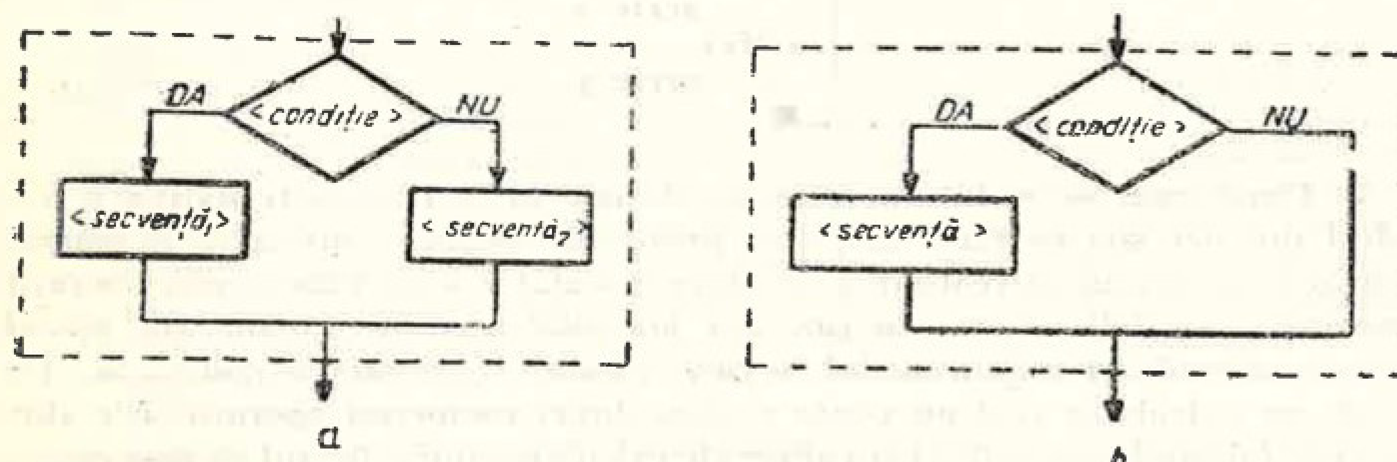
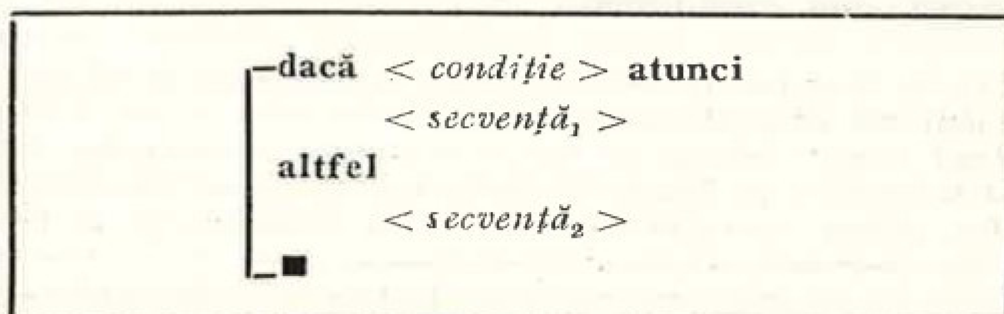


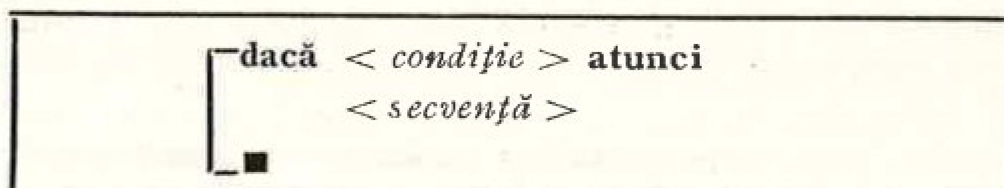
Fig. I.17

În LPS, prima variantă a raționamentului condițional o vom descrie cu ajutorul instrucțiunii condiționale sub forma :



```
┌dacă < condiție > atunci
│        < secvență₁ >
│altfel
│        < secvență₂ >
└■
```

iar varianta a doua, cu o alternativă nulă ca efect, astfel :

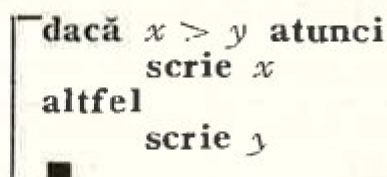


```
┌dacă < condiție > atunci
│        < secvență >
└■
```

În descrierile de mai sus < *condiție* > desemnează o expresie logică, deci o expresie care prin evaluare produce ca rezultat **adevărat** sau **fals** cum ar fi : $a > b$, $x = 0$, $alfa \leqslant 4 \vee beta - 5 < min$ în care ordinea de efectuare a operațiilor este cea cunoscută : întîi operațiile algebrice cu prioritățile respective, apoi operațiile relaționale și ultimele operațiile logice.

< *secvență* >, < *secvență*$_1$ >, < *secvență*$_2$ > desemnează secvențe de instrucțiuni LPS de orice fel, inclusiv instrucțiuni condiționale.

Revenind la exemplul nostru, cu ajutorul instrucțiunii condiționale putem descrie operația prin care scriem pe BE cea mai mare valoare dintre cele două valori aflate în celulele x și y, astfel



```
┌dacă x > y atunci
│        scrie x
│altfel
│        scrie y
└■
```

2. După cum se vede, operația de decizie între două alternative a rezultat din necesitatea rezolvării unei probleme concrete, suficient de simple pentru a ne dovedi că realizarea de către calculator a unei asemenea operații este indispensabilă. *Forma în care am prezentat operația provine din modul în care raționăm și nu din modul în care calculatoarele reale o realizează.* De altfel, un calculator real nu poate realiza direct asemenea operații. Ele sînt *simulate* folosind operațiile elementare ale calculatorului și faptul că programul este memorat.

În figura I.18 este prezentat un exemplu al modului în care se simulează pe un calculator real instrucţiunea condiţională.

Adrese din memoria program	Instrucţiuni din memoria program	Comentarii

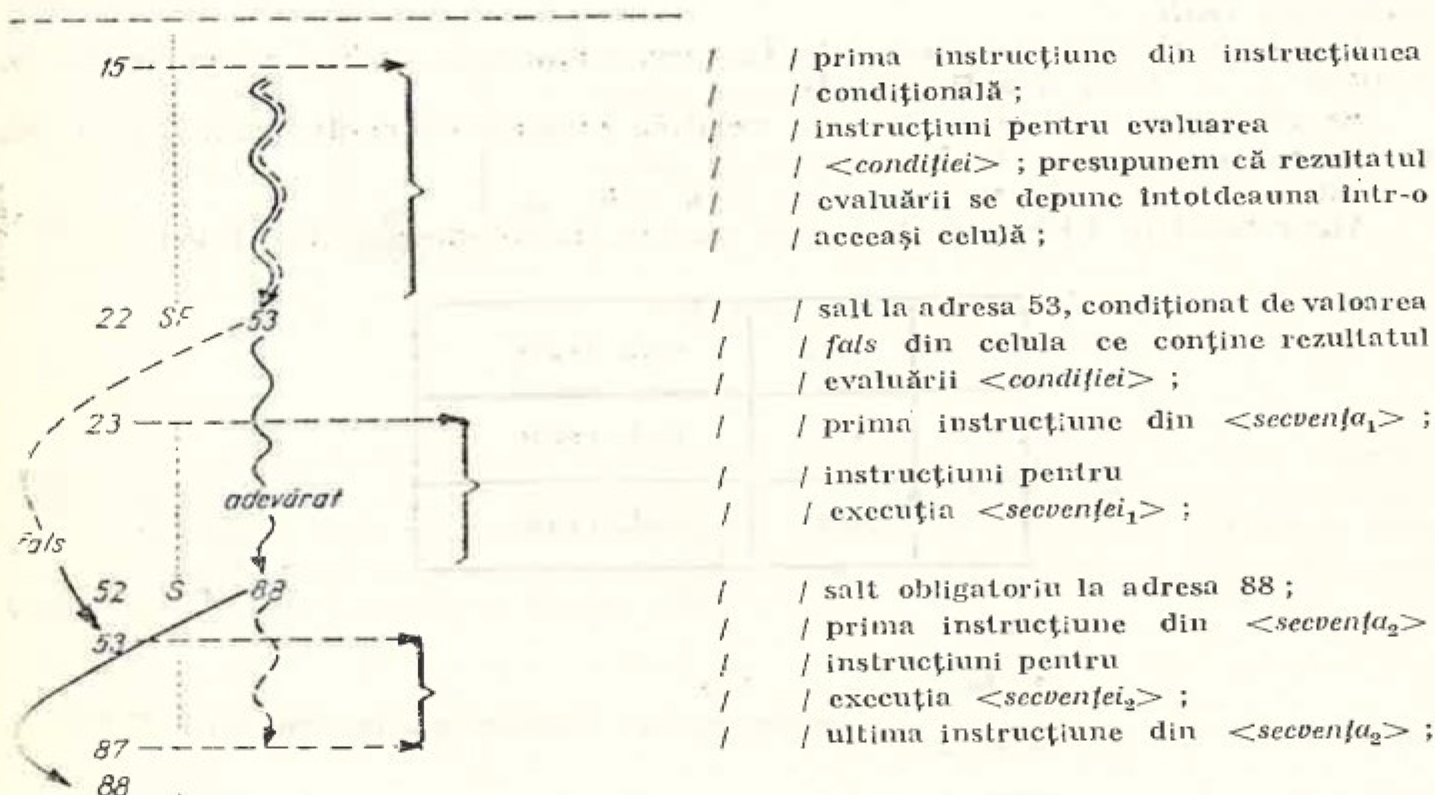


Fig. I.18

Instrucţiunea condiţională simulată în limbajul calculatorului se întinde între adresele 15 şi 87 ale memoriei programului. S-au pus în evidenţă instrucţiunea SF 53 („salt la fals") aflată la adresa 22 din memoria programului şi instrucţiunea S 88 („salt necondiţionat") aflată la adresa 52. S-a figurat cu linie continuă cazul execuţiei primei alternative şi cu linie întreruptă cazul execuţiei alternativei a doua.

Calculatorul C fiind imaginat de noi, ne putem permite să presupunem că el realizează dintr-o dată operaţia de decizie chiar dacă realizarea ei în cazul calculatoarelor reale constă de fapt din trei faze: a) evaluarea condiţiei, b) decizia de a urma o cale sau alta în execuţie şi c) execuţia secvenţei alese.

3. Din cele de mai sus rezultă caracterul deosebit al instrucţiunii condiţionale faţă de instrucţiunile de bază. Din punct de vedere al efectului instrucţiunii condiţionale asupra unui calculator, să observăm că operaţia declanşată de ea schimbă starea calculatorului doar prin modificările descrise în secvenţele de instrucţiuni cuprinse în cadrul instrucţiunii, evaluarea condiţiei neducînd la modificarea stării. Rolul operaţiei este de a conduce execuţia pe una din cele două căi în funcţie de anumite relaţii existente între variabilele din program. Această decizie se ia *dinamic*, la execuţia programului.

Asemenea operaţie (instrucţiune), care stabileşte o altă ordine de execuţie a instrucţiunilor decît cea a scrierii acestora, se mai numeşte *operaţie (instrucţiune) de control* (al execuţiei).

Conţinînd la rîndul ei alte instrucţiuni (datorită secvenţelor incluse în sintaxa instrucţiunii), instrucţiunea condiţională este un exemplu de instrucţiune structurată. Aşa cum s-a mai arătat, instrucţiunile incluse pot fi chiar

instrucțiuni condiționale. Relația de includere este pusă în evidență prin scrierea decalată spre dreapta a secvențelor față de cuvintele **dacă** și **altfel.**

Algoritmul 2

Să se scrie algoritmul în LPS pentru rezolvarea ecuației de gradul I cu coeficienți reali.

Date inițiale : două valori reale. Le vom memora în cadrul programului în celulele a, b. Deci $a \in \mathbf{R}$, $b \in \mathbf{R}$.

Rezultate : o valoare reală care verifică ecuația cu coeficienții a și b. Să o notăm x. Deci : $x \in \mathbf{R}$.

Între a, b și x avem relația $ax + b = 0$.

Algoritmul în LPS pare la prima vedere foarte simplu (fig. I.19).

a	v.s.	val. reale
b	v.s.	val. reale
x	v.s.	val. reale

citește a, b

atribuie $x \leftarrow -b/a$

scrie x

stop

Fig. I.19

Să observăm că pentru $a = 0$, caz posibil deoarece $a \in \mathbf{R}$, ecuația devine

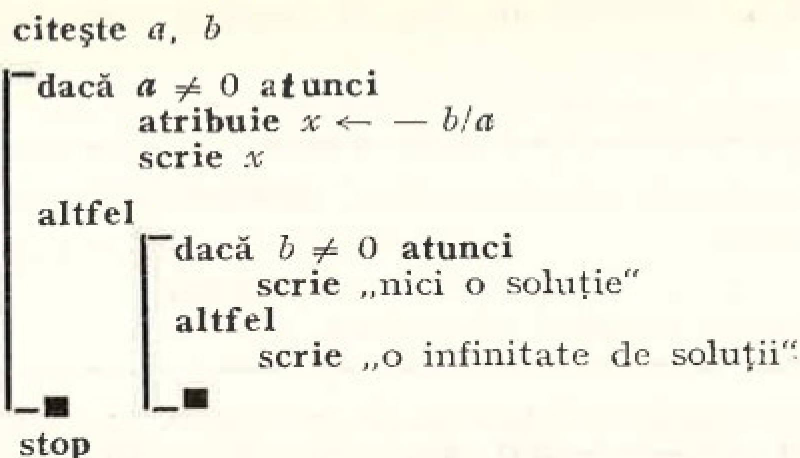
$$0 \cdot x + b = 0.$$

Relația este satisfăcută de orice $x \in \mathbf{R}$ dacă $b = 0$, dar niciodată satisfăcută de vreun $x \in \mathbf{R}$ dacă $b \neq 0$. De altfel, chiar împărțirea b/a trebuie să ne pună pe gînduri deoarece nu poate fi executată de calculator dacă $a = 0$ (împărțire cu zero).

Cazul $a \neq 0$ se poate rezolva ca în figura I.19. În celelalte două cazuri : $a = 0$ și $b \neq 0$ și respectiv $a = 0$ și $b = 0$, calculatorul nu poate realiza înscrierea rezultatului printr-o singură valoare deoarece avem fie nici o valoare, fie o infinitate de valori. În asemenea cazuri se obișnuiește să se tipărească pe BE un text (șir de caractere) din care să reiasă situația respectivă.

Algoritmul 2 în noua versiune, îmbunătățită prin sesizarea tuturor cazurilor, este prezentat în figura I.20.

a	v.s.	val. reale
b	v.s.	val. reale
x	v.s.	val. reale



```
citeşte a, b
┌dacă a ≠ 0 atunci
│      atribuie x ← — b/a
│      scrie x
│altfel
│      ┌dacă b ≠ 0 atunci
│      │      scrie „nici o soluţie"
│      │altfel
│      │      scrie „o infinitate de soluţii"
└■     └■
stop
```

Fig. I.20

Observaţie

În instrucţiunile de scriere s-a utilizat în loc de variabilă un text cuprins între ghilimele. Acest text poate fi privit ca o constantă de tip şir de caractere (aşa cum 5 este constantă întreagă, 1.738 este constantă reală sau **fals** este constantă logică). Ea se înscrie în celula curentă a benzii BE exact sub forma şirului de caractere dintre ghilimele.

I.3.2. Instrucţiuni de ciclare cu condiţie

1. Să ne reamintim metoda de determinare a celui mai mare divizor comun a două numere numită şi *algoritmul lui Euclid* şi să o aplicăm pentru cazul particular al numerelor pozitive 306 şi 119.

1° se împarte primul număr la al doilea determinînd restul : 306 = 2·119+68→rest = 68

2° se iau : al doilea din cele două numere şi restul : 119 şi 68

3° se împarte primul număr la al doilea determinînd restul : 119 = 1·68+51→rest = 51

4° se iau : al doilea din cele două numere şi restul : 68 şi 51

5° se împarte primul număr la al doilea determinînd restul : 68 = 1·51+17→rest = 17

6° se iau : al doilea din cele două numere şi restul : 51 şi 17

7° se împarte primul număr la al doilea determinînd restul : 51 = 3·17+0→rest = 0

8° se ia ca rezultat restul precedent (al doilea număr) : 17.

Să observăm că :

I) Operaţiile 1°, 3°, 5° şi 7° sînt una şi aceeaşi operaţie aplicată unor valori diferite. În acest fel primele 6 operaţii ale calculului constau din repetarea de 3 ori a aceleiaşi perechi de operaţii.

II) Numărul de repetări nu era previzibil înaintea calculului, iar aplicînd algoritmul pentru altă pereche de numere poate să rezulte un alt număr de repetări. De exemplu, pentru 1323 şi 875 este necesară repetarea de 5 ori a operaţiei de determinare a restului. Deci, modul de descriere folosit mai sus conduce la secvenţe diferite de operaţii dacă este aplicat la perechi diferite de numere. Ceea ce dorim este să descriem într-un singur program (deci într-un text finit) repetarea de mai multe ori a aceleiaşi secvenţe de operaţii, numărul de repetări fiind oricît de mare, dependent de îndeplinirea unei condiţii.

Se constată relativ uşor că descrierea algoritmului lui Euclid în termenii instrucţiunilor învăţate pînă acum este imposibilă. Vom aborda o altă cale prin care se va impune ca necesară o operaţie nouă pentru calculatorul C şi deci o nouă instrucţiune în LPS.

2. Să încercăm să expunem în cuvinte modul în care raţionăm atunci cînd determinăm cu algoritmul lui Euclid c.m.m.d.c. a două numere.

Presupunem că am aplicat deja algoritmul lui Euclid pe cîteva cazuri şi ne putem permite generalizarea.

Se observă ușor că există un grup de operații care se repetă formînd un așa-numit *ciclu* și anume :

[— determinarea restului celor două numere
— pregătirea următoarelor două numere (A)

Deci am putea comanda calculatorului C execuția repetată a secvenței celor două operații prin :
„ciclează"

[— determinarea restului celor două numere
— pregătirea următoarelor două numere (B)

Numai că această repetare trebuie să se termine la un moment dat și anume cînd ultimul rest obținut este 0. Am putea să specificăm acest lucru în comanda execuției astfel
„ciclează"

[— determinarea restului celor două numere
— pregătirea următoarelor două numere (C)

„pînă cînd" restul este 0.
Cele două operații care se repetă pot fi descrise în LPS. Pentru aceasta vom plasa mai întîi valorile utilizate de aceste operații în memoria calculatorului. Ele sînt în primul rînd cele două numere ; să le memorăm în celulele m și n. Apoi, restul să-l memorăm în celula r.
Cu aceasta, operațiile din (C) se pot descrie în LPS astfel :

determinarea restului celor două numere

atribuie $r \leftarrow m - (m/n)*n$[1] (D)

pregătirea următoarelor două numere

atribuie $m \leftarrow n$ & $n \leftarrow r$

Reamintim utilizarea semnului / pentru împărțire. Aici operația de împărțire are loc în **Z.** De exemplu $17/5 = 3$, $7/9 = 0$ etc.
Rescriind forma (C) a prelucrării conform descrierii în LPS din (D) se obține :

„ciclează"

atribuie $r \leftarrow m - (m/n)*n$

atribuie $m \leftarrow n$&$n \leftarrow r$

„pînă cînd" $r = 0$

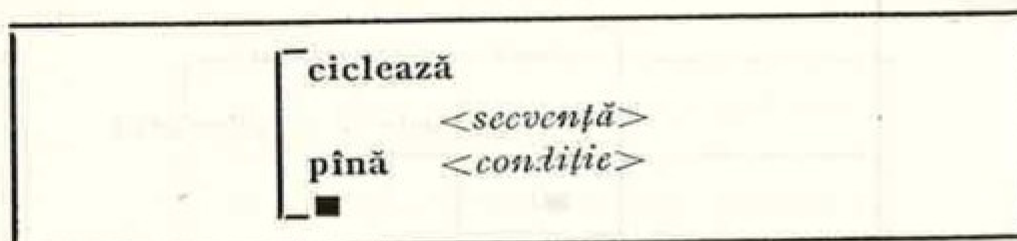
Intuitiv, cunoaștem semnificația operației determinate de cuvintele „ciclează"... „pînă cînd". Ea nu se poate descrie cu ajutorul instrucțiunilor LPS învățate. Pe de altă parte operația se întîlnește frecvent în raționamentele noastre. De aceea vom introduce o nouă instrucțiune structurată în LPS, *instrucțiunea de ciclare cu test final*, care să determine repetarea sub controlul unei condiții a execuției unei secvențe de instrucțiuni. Este deci un alt exemplu de instrucțiune de control al execuției.

---

[1] În condițiile folosirii unei funcții de forma *mod* (m, n) pentru restul modulo n al lui m, atribuirea se mai scrie și astfel :

atribuie $r \leftarrow$ *mod* (m, n)

Scrierea ei în LPS respectă forma de mai sus fixînd cuvintele care îi determină semnificația :



Semnificația instrucțiunii astfel scrise este echivalentă cu cea descrisă prin schema logică din figura I.21, *a*.

Secvența de instrucțiuni formează *corpul ciclului*.

Din execuția instrucțiunii de ciclare cu test final se observă o acumulare a efectelor execuțiilor succesive ale secvenței de instrucțiuni. Într-adevăr, executarea a n-a oară a secvenței de instrucțiuni ($n \geq 2$) se face plecînd din starea SPDS rezultată în urma celei de a $(n-1)$-a execuții a aceleiași secvențe. Această stare a calculatorului trebuie să fie diferită de stările din care au început execuțiile precedente. În caz contrar ciclarea este infinită. Astfel, se poate observa că dacă la un moment dat în șirul de execuții ale secvenței realizat pînă în acel moment se identifică două execuții ale secvenței care lasă sistemul SPDS într-o aceeași stare, execuțiile vor continua la infinit : condiția nu se va verifica niciodată. Același fenomen are loc dacă execuțiile secvenței nu schimbă nici una din variabilele din condiție.

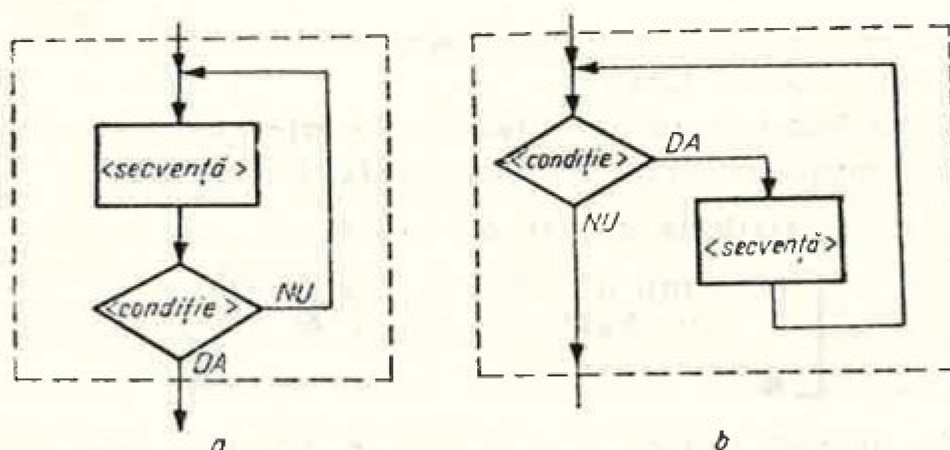
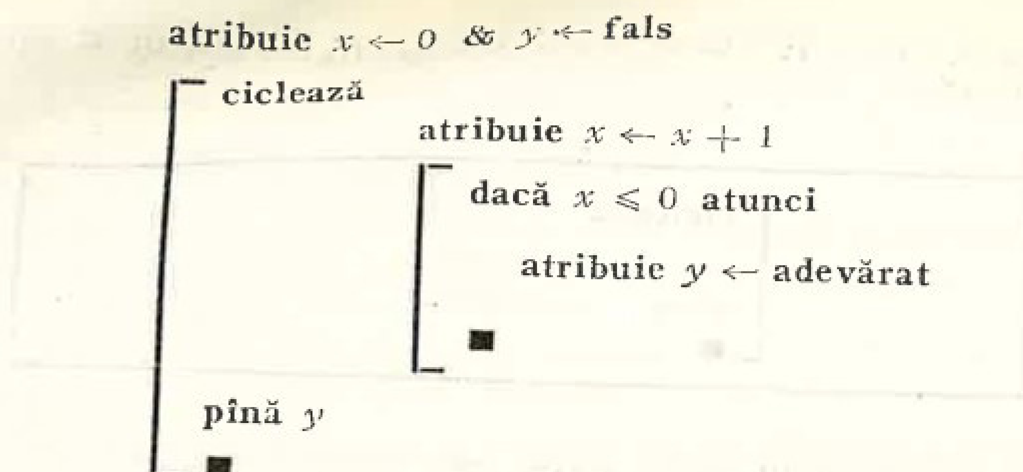


Fig. I.21

De exemplu :

x	v.s.	val. întregi
y	v.s.	val. logice

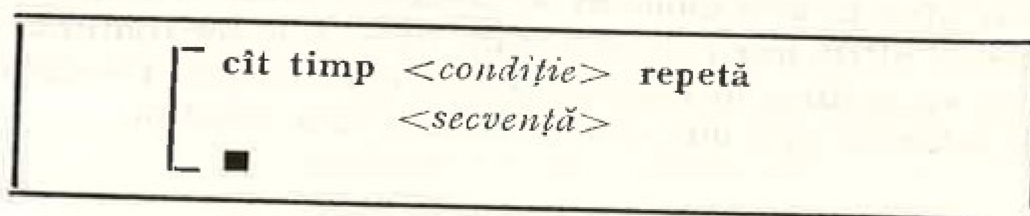


```
atribuie x ← 0 & y ← fals
┌ ciclează
│        atribuie x ← x + 1
│        ┌ dacă x ≤ 0 atunci
│        │     atribuie y ← adevărat
│        └ ■
│  pînă y
└ ■
```

Condiția de terminare a ciclării este formată din variabila y (al cărei tip este logic). Deci ciclarea se termină cînd y are valoarea *adevărată*. Acest lucru nu se va întîmpla însă niciodată, deoarece y își modifică valoarea **fals** doar dacă $x \leqslant 0$.

3. În unele prelucrări este utilă o altă instrucțiune de ciclare în care testul se face la început.

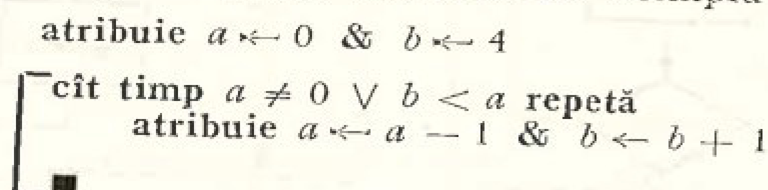
O vom scrie astfel :



```
┌ cît timp <condiție> repetă
│         <secvență>
└ ■
```

Semnificația *instrucțiunii cu test inițial* este echivalentă cu cea descrisă prin schema logică din figura I.21, *b*.

Se observă că înlănțuirea execuțiilor secvenței poate să nu aibă loc dacă, de la început, semnificația condiției este **fals.** De exemplu :



```
atribuie a ← 0 & b ← 4
┌ cît timp a ≠ 0 ∨ b < a repetă
│     atribuie a ← a − 1 & b ← b + 1
└ ■
```

Deoarece la început a este 0 și b este 4, condiția $a \neq 0 \vee b < a$ are valoarea **fals,** iar ciclarea nu are loc.

Algoritmul 3 (Algoritmul lui Euclid)

Să se descrie în LPS algoritmul pentru determinarea celui mai mare divizor comun a două numere întregi strict pozitive.

Date inițiale : două numere întregi strict pozitive aflate pe BI.

Rezultate : un număr întreg strict pozitiv care este c.m.m.d.c. al numerelor aflate pe BI.

Algoritmul în LPS pentru determinarea c.m.m.d.c. al celor două numere este prezentat în figura I.22. Celulele m și n folosesc pentru memorarea celor două numere aflate la un moment dat în discuție, iar celula r pentru memorarea restului.

m	v.s.	val. întregi pozitive
n	v.s.	val. întregi pozitive
r	v.s.	val. întregi pozitive
t	v.s.	val. întregi pozitive

```
citește m, n
┌ dacă m ≠ n atunci
│        ┌dacă m < n atunci
│        │       atribuie t ← m & m ← n & n ← t
│        └■
│        ┌ciclează
│        │   atribuie r ← m − (m/n)*n
│        │   atribuie m ← n & n ← r
│        │ pînă r = 0
│        └■
└■
scrie m
stop
```

Fig. I.22

Din aplicarea algoritmului lui Euclid se știe că împărțind numărul mai mare la cel mai mic calculul are mai puține operații decît în cazul împărțirii numărului mic la cel mare. Este deci de dorit ca în m să se afle numărul mai mare și în n cel mai mic. De aceea, înainte de ciclare trebuie să ne asigurăm de acest lucru printr-o comparare a valorilor din m și n după care dacă în m valoarea este mai mică decît cea din n, vom schimba valorile între cele două celule. Această schimbare necesită trei atribuiri și o celulă suplimentară t în care se „salvează" una din valori. Succesiunea transferurilor de date între cele trei celule m, n și t este prezentată în figura I.23.

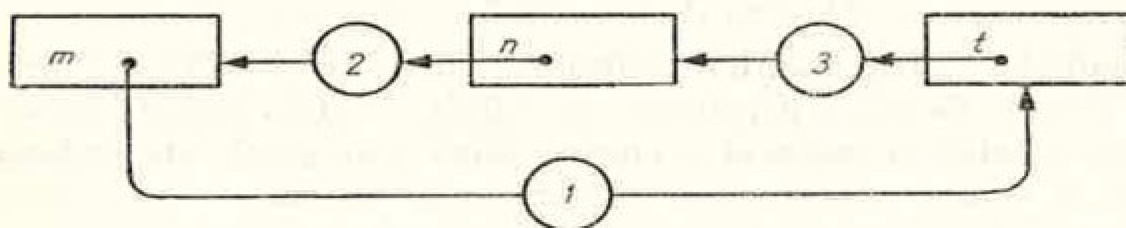


Fig. I.23

În fine, în figura I.22 se mai poate constata că s-au evitat calculele inutile pentru cazul $m = n$.

I.3.3. Utilizarea tablourilor în LPS

1. Ne propunem să scriem în LPS algoritmul pentru ordonarea descrescătoare a 5 numere reale aflate pe BI.

a) Fie, în ordine pe banda BI, următoarele 5 numere :

315,12 ; 173,5 ; 82,9 ; —15 ; —33,2.

Evident, nu avem ce ordona, ele fiind ordonate, deoarece :

$$315{,}12 \geqslant 173{,}5 \geqslant 82{,}9 \geqslant -15 \geqslant -33{,}2.$$

Deci le putem înscrie exact în această ordine pe banda BE, după ce le-am plasat temporar în 5 celule din memorie (*Drumul de la BI spre BE trece în mod obligatoriu prin memorie*).

b) Fie în ordine următoarele 5 numere pe banda BI :

315,12 ; 173,5 ; —33,2 ; 82,9 ; —15.

Ele nu mai pot fi transferate în această ordine pe BE, deoarece ordinea de pe BI nu mai corespunde ordinii descrescătoare a celor 5 numere :

$$-33{,}2 < 82{,}9.$$

Soluția este să le păstrăm deocamdată în memorie în vederea unor prelucrări avînd ca scop ordonarea valorilor. Odată ordonate, le vom înscrie în ordinea respectivă pe banda BE. În figura I.24 sînt prezentate cele 5 ce-

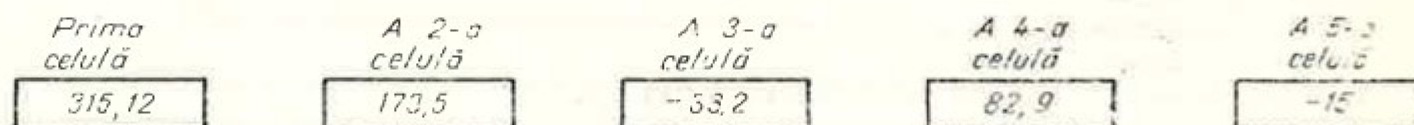


Fig. I.24

lule din memorie ce conțin valorile citite. Celulele sînt și ele ordonate datorită unei numerotări : prima, a doua, . . . , a cincea. Scopul nostru este să permutăm valorile în celule, astfel încît în prima celulă să avem cea mai mare valoare, în a doua valoarea următoare etc. În cazul din figură, inspectînd de la stînga la dreapta celulele, o primă schimbare ar fi între celula a treia și a 4-a. Obținem șirul :

315,12 ; 173,5 ; 82,9 ; —33,2 ; —15.

După modificare șirul rămîne neordonat deoarece : $-33{,}2 < -15$.
Din nou trebuie făcută o permutare de valori, de data aceasta schimbînd valorile între celulele a patra și a cincea, după care șirul este ordonat.

c) Fie în ordine următoarele 5 numere pe bandă :

82,9 ; —33,2 ; 315,12 ; 173,5 ; —15.

Aplicînd compararea două cîte două, de la stînga la dreapta, a numerelor memorate în această ordine descoperim că după $82{,}9 \geqslant -33{,}2$, avem $-33{,}2 <$

$< 315{,}12$, deci locul lui 315,12 este în fața lui —33,2. Schimbăm prin urmare valorile între celulele a doua și a treia, ceea ce ne conduce la șirul :

82,9 ; 315,12 ; —33,2 ; 173,5 ; —15.

Dacă verificăm în continuare perechile, vedem că —33,2 trebuie să-și schimbe locul cu 173,5 și apoi cu —15, șirul devenind :

82,9 ; 315,12 ; 173,5 ; —15 ; —33,2.

Rezultatul constă în plasarea celui mai mic număr în ultima celulă. Șirul nu este ordonat, deoarece, reluînd compararea, constatăm că $82{,}9 < 315{,}12$, deci trebuie schimbate valorile primelor două celule ș.a.m.d.

Rezumînd, putem spune că ordonarea are loc inspectînd valorile aflate în perechi de celule consecutive în încercarea de a le ordona corect : în prima celulă valoarea mai mare, în a doua celulă valoarea mai mică. Dacă valorile lor sînt deja ordonate, trecem la următoarea pereche de celule, dacă nu, permutăm cele două valori. Ordonarea se termină cînd, parcurgînd încă o dată șirul celulelor, constatăm că nu mai este necesară nici o permutare.

2. Să observăm că în cele de mai sus am considerat de la început celulele (nu valorile !) ordonate. Deci am convenit de la început : aceasta este celula care în final va conține cel mai mare număr, ea este prima, aceasta este celula ce va conține în final următorul număr, ea este a doua ș.a.m.d. Să presupunem că le-am și denumit în vederea scrierii programului ; în ordine sînt : *a*, *b*, *c*, *d*, *e*. Atunci o trecere prin șirul valorilor va fi înscrisă în LPS astfel :

dacă $a < b$ **atunci**
  atribuie $t \leftarrow a$ & $a \leftarrow b$ & $b \leftarrow t$
■

dacă $b < c$ **atunci**
  atribuie $t \leftarrow b$ & $b \leftarrow c$ & $c \leftarrow t$
■

dacă $c < d$ **atunci**
  atribuie $t \leftarrow c$ & $c \leftarrow d$ & $d \leftarrow t$
■

dacă $d < e$ **atunci**
  atribuie $t \leftarrow d$ & $d \leftarrow e$ & $e \leftarrow t$
■

Se constată că sînt necesare 4 instrucțiuni condiționale pentru a trece în revistă cele 5 numere. Cu 100 de numere ar fi necesare 99 instrucțiuni condiționale, iar cele 100 celule distincte le-am denumi probabil : *a*, *b*, . . . , *z*, *ab*, *bc*, *cd*, ... Dar dacă ar fi de ordonat 10 000 de numere, cum am denumi celulele și cîte instrucțiuni condiționale ar fi necesare ? E clar că pentru a descrie metoda de ordonare a unei mulțimi oarecare de numere trebuie încercat altfel.

3. Vom pleca de la observația că cele 4 instrucțiuni condiționale de mai sus prelucrează la fel o pereche de celule, pereche care însă se schimbă de la un caz la altul. Într-adevăr, cu excepția numelor celulelor, cele 4 instrucțiuni sînt identice. Nu s-ar putea să scriem prelucrarea realizată o singură dată, dar s-o executăm de patru ori schimbînd la fiecare execuție, printr-un anumit

procedeu, numele celulelor asupra cărora lucrăm ? Pentru aceasta vom considera cele 5 celule ca un ansamblu căruia îi dăm un nume, iar în cadrul ansamblului vom numerota celulele de la 1 la 5 în vederea deosebirii lor (fig. I.25).

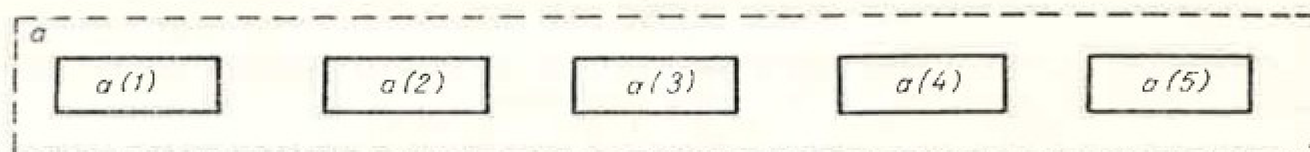


Fig. I.25

Reamintim că un asemenea ansamblu formează, conform celor discutate în I.2.4, un *tablou*, iar o celulă din ansamblu se numește *element de tablou*. Pentru a identifica tabloul folosim un nume ca și în cazul variabilelor, iar pentru a identifica un element de tablou folosim numele tabloului din care face parte, precum și numărul de ordine. Acest număr de ordine poartă numele de *indice*. În descrierea algoritmilor în LPS, în tabelul asociat algoritmului specificăm faptul că este vorba de un tablou, numărul dimensiunilor tabloului și domeniul valorilor indicilor pentru fiecare dimensiune. În referirile la un element de tablou vom folosi notarea indicelui între paranteze. Pentru $a(3)$ este vorba de al treilea element al unui tablou numit a, care, evident, va avea cel puțin 3 elemente. Uneori valoarea indicelui poate fi memorată într-o altă celulă din memorie. De exemplu, $tab(i)$ unde i este o celulă din memorie ce conține o valoare întreagă pozitivă. Evident, valoarea lui i trebuie să fie mai mică sau cel mult egală cu numărul de celule ale tabloului tab.

În fine, indicele poate fi dat și de o expresie oarecare ce are ca valoare un număr întreg pozitiv. De exemplu : $a(i-j-1)$ unde i și j sînt celule ce conțin valori întregi. Deoarece o asemenea notație identifică, ca și în cazul unei variabile, o celulă din memorie, vom face distincție între *variabilă indexată* (notația pentru un element de tablou) și *variabilă simplă* (notația folosită pînă acum pentru o celulă singulară din memorie).

Revenind la ordonarea numerelor vom scrie operația condițională care se repetă, sub forma :

```
┌ dacă a(i) < a(i + 1) atunci
│    ┌ atribuie t ← a(i)
│    │        & a(i) ← a(i + 1)
│    │        & a(i + 1) ← t
│    └ ■
└ ■
```

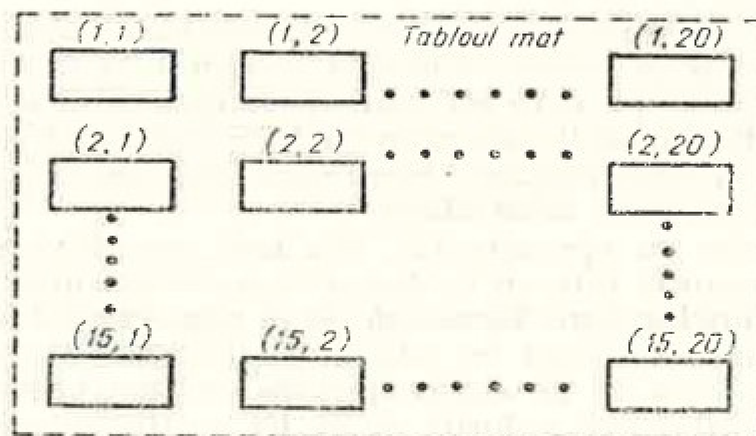
și vom obliga pe i să ia valori de la 1 la 4 astfel :

```
  atribuie i ← 1
┌ cît timp i ⩽ 4 repetă
│    ┌ dacă a(i) < a(i + 1) atunci
│    │    ┌ atribuie t ← a(i)
│    │    │        & a(i) ← a(i + 1)
│    │    │        & a(i + 1) ← t
│    │    └ ■
│    └ ■
│    atribuie i ← i + 1
└ ■
```

4. Conform celor prezentate în I.2.4 un tablou ca cel de mai sus este similar unui vector : mulțime de valori ordonate după un singur criteriu. Putem extinde aceste considerații și pentru matrice, deci mulțimi de valori ordonate după două criterii.

Să ne închipuim o zonă din memoria M cum este cea din figura I.26, **să-i dăm** un **nume** : *mat* **și să localizăm** o **celulă** din această zonă prin doi **indici : indicele liniei și, respectiv, indicele** coloanei. Astfel, *mat*(2,5) este numele **celulei din tabloul** *mat* **aflată** pe linia a 2-a și coloana a 5-a.

Fig. I.26



Valorile aflate în celulele zonei *mat* sînt toate de același tip și formează de fapt un tablou de valori (același cu cel din exemplul din I.2.4) ce poate fi asimilat cu o matrice.

Observație. Plasarea în memorie a celulelor în cazul calculatoarelor reale nu corespunde exact situației din figura I.26, deoarece memoria acestora este lineară. De aceea descrierea cu mai mulți indici a elementelor de tablou trebuie linearizată de către compilatorul limbajului de programare.

În fine, putem să extindem considerațiile la mulțimi de valori ordonate după mai multe criterii, cărora le corespund zone de memorie, tablouri, în care celulele sînt localizate folosind mai mult de doi indici, deci cu mai mult de două dimensiuni. De exemplu, *tab* (3, 5, 8) este o celulă dintr-un tablou în care accesul se face pe baza a 3 indici : indicele de linie, cel de coloană și cel de plan (dacă ne imaginăm această zonă în spațiu).

Algoritmul 4 (ordonarea numerelor)

Să se scrie algoritmul în LPS pentru înscrierea pe BE în ordine descrescătoare a unui șir de valori reale aflate pe banda BI. Numărul acestor valori (număr întreg pozitiv) se află în prima celulă a benzii BI. El este mai mic decît 100.

Date inițiale : un număr întreg pozitiv și apoi un șir de numere reale de lungime egală cu numărul întreg.

Rezultate : un șir de numere reale, aceleași cu cele citite, dar în ordine descrescătoare a valorilor.

Procedeul de ordonare a fost prezentat mai sus pentru cazul a 5 numere. Vom „construi" descrierea algoritmului în LPS pentru un caz mai general încercînd să descifrăm pas cu pas ceea ce trebuie făcut pentru a realiza cu calculatorul C ordonarea cerută. Metoda de construire a algoritmului este deosebit de importantă, de aceea recomandăm abordarea ei cu atenție.

Din cele prezentate a reieșit clar că valorile trebuie memorate în M, și aici, în memorie, ele trebuie prelucrate de către dispozitivul de prelucrare DP, după care se înscriu pe BE. Deci putem da pentru început descrierea prelucrării sub forma unei secvențe de patru operații de prelucrare :

```
(0)          | (1) * citirea valorilor de pe BI
             | (2) * ordonarea valorilor în memorie
             | (3) * înscrierea valorilor ordonate pe BE
             |     stop
```

Primele trei instrucțiuni nu sînt instrucțiuni LPS (asteriscul ce le precede arată tocmai acest lucru). Ele sînt descrieri în limbaj natural ale unor prelucrări complexe pe care nu știm deocamdată să le descriem exact în LPS. În final va trebui să le descriem în LPS, care este pentru noi *forma standard* de descriere a algoritmilor. Asemenea descrieri de operații precedate de * se numesc *enunțuri nestandard*.

Să începem cu operația (2), cea mai complexă dintre cele trei. Valorile se află în memorie într-un tablou a cu o dimensiune, pe scurt, într-un vector. Numărul celulelor care formează zona respectivă trebuie cunoscut de la început pentru a fi trecut în tabelul atributelor. Dacă îl luăm 100, este suficient pentru ceea ce ne-am propus în problemă. Numărul efectiv de celule în care s-au memorat valorile de pe BI se află memorat și el într-o celulă n.

Ceea ce știm de la început despre operația (2) este că realizează inspectarea repetată a șirului valorilor în efortul de a le ordona, pînă cînd, într-adevăr, le ordonează. Adică :

```
(2)'         ┌ ciclează
             |          (2.1) * inspectează și permută într-o trecere șirul
             | pînă     (2.2) * șirul este ordonat
             └ ■
```

(2)' este o descriere echivalentă cu (2) din punct de vedere al intențiilor noastre. Deosebirea constă în faptul că (2)' este o descriere mai detaliată decît (2) a prelucrării care, în final, va trebui să fie descrisă în LPS. În (2)' găsim două enunțuri nestandard dintre care (2.1) este o acțiune (ea are un efect modificator asupra stării calculatorului), în timp ce (2.2) este o condiție. Vom deosebi prin ' (apostrof) descrierile echivalente ale aceleiași operații.

Operația (2.1) poate fi scrisă ca o repetare de $n - 1$ ori (prin n înțelegem aici valoarea din celula n) a comparației valorilor din două celule consecutive :

```
(2.1)'          atribuie i ← 1
               ┌ cît timp i ≤ n − 1 repetă
               |            (2.1.1) * compară două celule consecutive
               |            atribuie i ← i + 1
               └ ■
```

În fine, operația (2.1.1) este cea identificată în considerațiile din paragraful precedent, adică :

```
(2.1.1)'              ┌ dacă a(i) < a(i + 1) atunci
                      |      ┌ atribuie t ← a(i)
                      |      |  & a(i) ← a(i + 1)
                      |      |  & a(i + 1) ← t
                      |      └ ■
                      └ ■
```

ntorcîndu-ne la (2)′ să observăm că atenția acordată exclusiv lui (2.1)′ a făcut să fie dificilă descrierea în LPS a ceea ce înseamnă (2.2). De aceea va trebui să revenim în (2.1)′ și (2.1.1)′, să le modificăm, pentru a putea descrie în LPS operația (2.2). Să acordăm mai multă atenție acestei reveniri pentru a evita repetarea greșelii pe viitor.

Este clar că (2.2) va folosi informații puse la dispoziția sa de către (2.1). Deci există una sau mai multe celule din memorie ce vor găzdui aceste informații transmițîndu-le de la operația (2.1) la operația (2.2). Informația necesară în (2.2) este răspunsul la întrebarea : la o trecere prin șirul de valori s-a realizat vreo permutare a valorilor sau nu ? În primul caz șirul nu este ordonat, în al doilea caz el este ordonat. Informația este cu două valori : da sau nu, 0 sau 1, **adevărat** sau **fals.** Să considerăm ultima codificare și o celulă *sem* care „semnalizează" prin **adevărat** dacă nu s-a făcut nici o permutare și prin **fals** dacă a existat măcar o permutare. Celula este actualizată (modificată) de către (2.1) și testată de (2.2). Vom scrie acest lucru astfel :

```
(2)''  ┌ ciclează
       │        (2.1) * inspectează și permută într-o trecere șirul (n, a ;
       │                a, sem)
       │ pînă (2.2) * șirul este ordonat (sem) ;
       └ ■
```

În (2)′′ apar în plus față de (2)′ pentru fiecare enunț nestandard niște argumente între paranteze. Argumentele indică sub forma unor liste de nume simbolice celulele din memorie care afectează sau care sînt afectate de operația respectivă. Prima categorie de celule formează lista de intrare, iar a doua, lista de ieșire. Cele două liste sînt separate prin ;. După cum se vede, (2.1) are nevoie de celulele *n* și *a* (toate celulele tabloului), informațiile aflate în ele determinînd acțiunea, efectul lui (2.1). În acelaș timp (2.1) poate modifica conținutul celulelor *a* și *sem*. Spunem că cele două liste formează *„interfața"* operației respective cu celelalte operații. Astfel, *sem* este celula ce formează „interfața" dintre (2.1) și (2.2) și am văzut că am fost obligați s-o folosim tocmai pentru transferul informațiilor de la (2.1) la (2.2).

Specificarea variabilelor interfeței unei operații nestandard cu celelalte operații din program este utilă în faza de proiectare a programelor, mai ales în cazul programelor complexe, de mari dimensiuni.

Să reluăm detalierea operației (2) adăugînd interfețele operațiilor nestandard.

```
         atribuie sem ← adevărat
(2.1)''  atribuie i ← 1
       ┌ cît timp i ≤ n − 1 repetă
       │      (2.1.1) * compară două celule consecutive (i, a ; a, sem)
       │      atribuie i ← i + 1
       └ ■
```

Să observăm că **atribuie** *sem* ← **adevărat** echivalează cu asumarea unei ipoteze : șirul este ordonat. Urmează ca (2.1.1) să ne contrazică sau nu prin efectul execuției ei asupra lui *sem*.

(2.1.1)''

```
┌ dacă a(i) < a(i + 1) atunci
│     ┌ atribuie t ← a(i)
│     │          & a(i) ← a(i + 1)
│     │          & a(i + 1) ← t
│     └ ■
│     atribuie sem ← fals
└ ■
```

Iată cum (2.1.1)'' dezvăluie complet rolul lui *sem* : acela de „semafor" ce dă „liber" sau nu operației de transcriere a valorilor pe BE. Într-adevăr, dacă $a(i) < a(i + 1)$ urmează o schimbare a valorilor acestor celule în acest caz și o atribuire a valorii **fals** lui *sem*, ceea ce de altfel urmăream. Celula *sem*, după cum se vede din (2.1)'', nu mai capătă valoare **adevărat** în cursul parcurgerii curente a șirului, ci cel mult i se mai confirmă de cîteva ori valoarea **fals.**

Exercițiu: Verificați acest lucru executînd secvența (2.1)'' pentru șirul : 1,7 ; —2,25 ; 10,3.

Ne rămîne să scriem cine este în noile condiții (2.2), ceea ce este foarte simplu :

(2.2)' *sem*

Revenind la prima formă a programului, să o rescriem și pe aceasta semnalînd interfețele :

(0)' (1) * citirea valorilor de pe BI (; n, a)[1]
(2) * ordonarea valorilor în memorie (n, a ; a)
(3) * înscrierea valorilor ordonate pe BE (n, a ;)
stop

Ne vom îndrepta atenția spre operațiile (1) și (3). (1) indică citirea unui număr de valori de pe BI, număr pe care nu îl cunoaștem acum, cînd scriem programul, deci lista identificatorilor din instrucțiunea de citire are o lungime necunoscută ce depinde de prima valoare citită. În plus, valorile citite se vor depune în celule aparținînd tabloului a. Dacă ar fi să scriem numele acestor celule, ar trebui să facem o listă, astfel :

$$a(1),\ a(2),\ a(3),\ \ldots,\ a(n)$$

unde n este numele unei celule din memorie în care se află numărul de valori de pe BI.

Deoarece specificația nu este precisă (formularea cu... nu este admisibilă), va trebui să specificăm într-o manieră riguroasă modul în care se determină valorile consecutive ale indicilor. Vom introduce notația $(a(i),\ i = 1,\ n)$ pentru lista de mai sus. Se observă că între paranteze se găsește termenul general $a(i)$ al tabloului a și expresia $i = 1,\ n$, care, prin convenție, ne arată cum ia valori celula i : de la 1 pînă la valoarea din n crescînd mereu cu 1 valoarea precedentă.

Deci putem detalia operația (1) astfel :

(1)' **citește** n

citește $(a(i),\ i = 1,\ n)$

---

[1] Se observă din poziționarea semnului ; lipsa listei de intrare în operația (1) iar în operația (3) lipsa listei de ieșire.

La fel pentru operația (3):

(3)' **scrie** $(a(i),\ i = 1,\ n)$

Înlocuind succesiv formele finale, în LPS, ale operațiilor nestandard și adăugînd declarațiile atributelor variabilelor utilizate obținem algoritmul din figura I.27.

De multe ori este util să menționăm, dacă este posibil, un domeniu mai restrîns decît mulțimile **Z**, **R**, pentru variabilele utilizate în program. Acest lucru l-am făcut în tabelul din figura I.27, pentru variabila n. Domeniul specificat prin expresia [1, 100] este cel al tuturor valorilor întregi între 1 și 100.

n	var. simplă	valori întregi [1, 100]
a	tablou 1 dimens. (100)	valori reale
i	var. simplă	valori întregi
t	var. simplă	valori reale
sem	var. simplă	valori logice

```
citește n
citește (a(i), i = 1, n)
┌ ciclează
│     atribuie sem ← adevărat
│     atribuie i ← 1
│     ┌ cît timp i ⩽ n − 1 repetă
│     │     ┌ dacă a(i) < a(i + 1) atunci
│     │     │ atribuie t ← a(i) & a(i) ← a(i + 1) & a(i + 1) ← t
│     │     │ atribuie sem ← fals
│     │     └ ■
│     │   atribuie i ← i + 1
│     └ ■
│  pînă sem
└ ■
scrie (a(i), i = 1, n)
stop
```

Fig. I.27

O astfel de construire pas cu pas a algoritmului unei probleme prin detalierea componentelor nestandard este o modalitate tot mai frecvent utilizată în proiectarea sistematică a programelor. Se numește *proiectare descendentă pas cu pas a programului*.

I.3.4. Instrucțiunea de ciclare cu contor

1. Să reluăm secvența (2.1)'' din exemplul precedent, de ordonare a unor valori reale, și să ne amintim rolul variabilei simple i în prelucrarea considerată. În afară de faptul că ia valori întregi, să mai observăm că aceste valori

se află între 1 și valoarea din n. Într-adevăr, i ia valoarea 1 înainte de a intra în ciclul cu test inițial, iar în interiorul ciclului, la sfîrșitul acestuia, valoarea din i crește cu 1 pînă în momentul în care $i > n - 1$ (adică $i = n$). În acest moment repetarea execuției instrucțiunilor din ciclu se oprește. Putem spune că i numără astfel execuțiile secvenței din ciclu sau le *contorizează*. De aici denumirea de *contor* dată unei asemenea variabile.

Prelucrările cu contor sînt frecvente în aplicații și de aceea se pot întîlni în multe limbaje de programare instrucțiuni speciale pentru descrierea lor. Vom introduce și noi o asemenea instrucțiune în LPS, observînd totuși că ea poate fi „simulată" cu ajutorul instrucțiunilor de ciclare cu test inițial sau final.

Instrucțiunea de ciclare cu contor se scrie în LPS astfel:

pentru *<var>* = *<vinit>*, *<vfin>*, *<vpas>* **execută**
    <secvență>
■

unde: *<var>* este contorul, o variabilă simplă de tip întreg, iar ***<vinit>***, *<vfin>*, *<vpas>* sînt numere sau variabile simple întregi. Dacă ***<vpas>*** este 1, el poate lipsi împreună cu virgula ce-l precede.

Efectul acestei instrucțiuni este dat de schema logică din figura I.28.

2. Să observăm o deosebire importantă între ciclarea cu contor și cea cu condiție. Chiar dacă în momentul în care se scrie programul se poate să nu se știe de cîte ori va fi executată secvența, acest lucru devine cunoscut

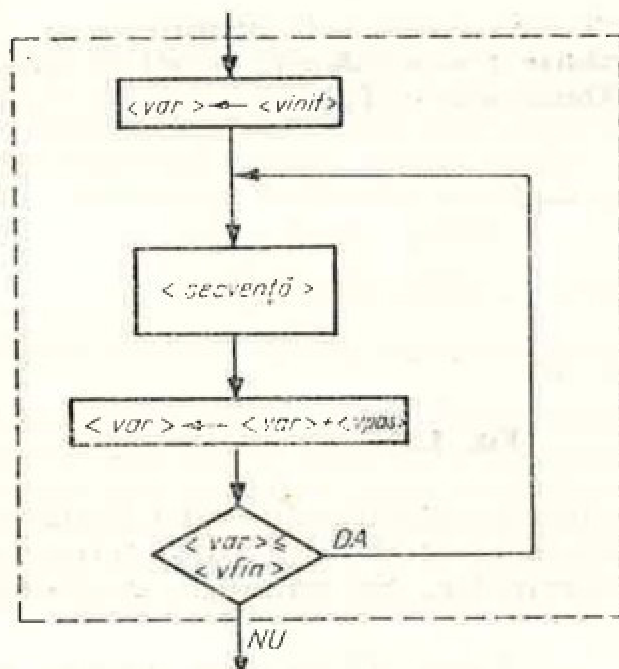


Fig. I.28

atunci cînd se începe execuția înlănțuită a secvenței rezultînd din calculul: $(vfin - vinit)/vpas + 1$. În ciclarea cu condiție sfîrșitul ciclării rezultă de obicei ca o consecință a efectului înlănțuirii respective. În aceste condiții nu se poate spune nici măcar la începutul execuției operației de ciclare cu condiție cînd se va termina ea.

Reluînd programul de ordonare din figura I.27 și refăcîndu-l pentru a folosi instrucțiunea de ciclare cu contor, rezultă programul din figura I.29.

n	var. simplă	valori întregi [1, 100]
a	tablou 1 dimens. (100)	valori reale
i	var. simplă (contor)	valori întregi
t	var. simplă	valori reale
sem	var. simplă	valori logice
m	var. simplă	valori întregi

```
citește n
citește (a(i), i = 1, n)
┌ciclează
│    atribuie sem ← adevărat
│    atribuie m ← n − 1
│ ┌pentru i = 1, m execută
│ │ ┌dacă a(i) < a(i + 1) atunci
│ │ │    atribuie t ← a(i) & a(i) ← a(i + 1) & a(i + 1) ← t
│ │ │    atribuie sem ← fals
│ │ └■
│ └■
│ pînă sem
└■
scrie (a(i), i = 1, n)
stop
```

Fig. I.29

De remarcat artificiul care asigură o variabilă simplă ca <*vfin*> în instrucțiunea de ciclare cu contor : s-a introdus o variabilă m care înaintea instrucțiunii de ciclare a fost inițializată cu valoarea unei expresii ce dă limita superioară a valorii contorului.

I.3.5. Probleme

1. Se dau două numere întregi.
   a) Să se scrie algoritmul în LPS ce inserie pe banda de ieșire cele două numere în ordine descrescătoare.
   b) Să se prezinte stările succesive ale SPDS în cursul execuției algoritmului.

2. Idem pentru trei numere întregi.

3. Să se scrie algoritmul în LPS pentru rezolvarea ecuației de gradul II cu coeficienți reali.
Indicație. Rădăcinile complexe conjugate se pot înscrie ca două valori reale : partea reală și partea imaginară. Pentru a nu confunda acest caz cu cel al rădăcinilor reale distincte este bine ca pe banda de ieșire să se tipărească și un text care să informeze utilizatorul asupra situației în care s-a terminat execuția.

4. Să se scrie algoritmul în LPS pentru calculul valorii funcţiei date pe intervale:

$$f(x) = \begin{cases} 1 - x \text{ dacă } x < -1, \\ \sqrt{1 - x^2} \text{ dacă } -1 \leqslant x \leqslant 1, \\ x - 1 \text{ dacă } x > 1. \end{cases}$$

5. Se dau pe banda de intrare două date calendaristice, fiecare formată din trei valori întregi strict pozitive reprezentînd ziua, luna şi respectiv anul. Prima dată este data la care se execută programul. A doua reprezintă data naşterii unei persoane. Să se scrie programul în LPS care înscrie pe banda de ieşire vîrsta în ani a persoanei respective. Dacă aceasta îşi serbează ziua de naştere în chiar ziua executării programului, se consideră că are o vîrstă cu un an mai mare decît cea din ziua precedentă.

6. Fie algoritmul din figura I.30

x	v.s.	v. reale
y	v.s.	v. reale
z	v.s.	v. reale

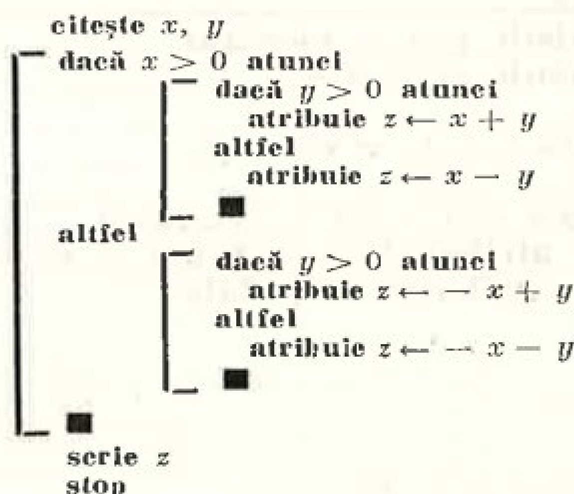


Fig. I.30

Să se arate că algoritmul verifică condiţia : $z = |x| + |y|$.

7. Fie algoritmul din figura I.31 reprezentat printr-o schemă logică în care m şi n sînt variabile întregi strict pozitive.
   a) Să se verifice că algoritmul este o variantă a algoritmului lui Euclid.
   b) Să se scrie algoritmul în LPS care descrie aceeaşi prelucrare.
8. Să se scrie algoritmul în LPS pentru calculul factorialului unui număr întreg pozitiv dat.
9. Fie ecuaţia $x \cdot e^x - 1 = 0$. Să se scrie algoritmul în LPS pentru determinarea, cu o eroare absolută maxim admisibilă dată, a unei soluţii aproximative a ecuaţiei. Se va utiliza metoda înjumătăţirii intervalului.

Indicaţii

— În intervalul [0, 1] se găseşte o singură soluţie exactă x_0. Într-adevăr, fie $f(x) = x \cdot e^x - 1$, atunci $f(0) = -1 < 0$ şi $f(1) = e - 1 > 0$, iar în acest interval $f(x)$ este continuă şi monoton crescătoare :

— Să notăm cu x_0 soluţia exactă, x_a cea aproximativă şi ε eroarea absolută maxim admisibilă. Fie $[a, b]$ un interval în care ştim că se găseşte soluţia exactă, atunci ;

$$(\forall x_a)(x_a \in [a, b])|a - b| < \varepsilon \Rightarrow |x_0 - x_a| < \varepsilon$$

Deci este suficient să găsim un interval $[a, b]$ astfel încît $|a - b| < \varepsilon$ pentru ca luînd orice punct x_a din acest interval să fim asiguraţi că $|x_0 - x_a| < \varepsilon$;

— Dacă $[a, b]$ este un interval în care se găseşte soluţia exactă atunci : sau $f(a) \leqslant 0$ şi $f(b) > 0$, sau $f(a) < 0$ şi $f(b) \geqslant 0$.

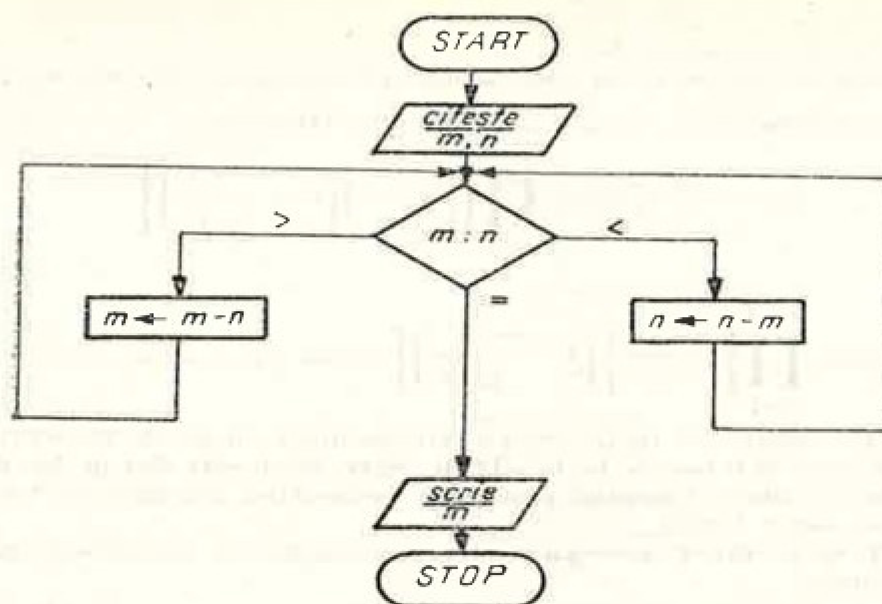


Fig. I.31

10. Să se scrie algoritmul în LPS pentru calculul rădăcinii de un ordin întreg pozitiv dat dintr-un număr real de asemenea dat. Rădăcina se calculează cu o eroare absolută dată (valoare reală).

Indicație

Se poate folosi metoda lui Newton de calcul a rădăcinii de ordinul p dintr-o valoare a (adică $\sqrt[p]{a}$). Conform acestei metode, şirul cu termenul general $x_n = \frac{1}{p}\left[(p-1)x_{n-1} + \frac{a}{(x_{n-1})^{p-1}}\right]$ converge spre $\sqrt[p]{a}$. Numărul rădăcinilor pe care le putem obține folosind acest şir depinde de semnul lui a şi de paritatea lui p astfel:

	$a>0$	$a<0$
p par	2 răd. $a^{1/p}, -a^{1/p}$	nici o răd.
p impar	1 răd. $a^{1/p}$	1 răd. $-\lvert a\rvert^{1/p}$

Alegerea lui x_0 are de asemenea importanță pentru determinarea rădăcinii astfel:

	p par, $a>0$	p impar, $a>0$	p impar, $a<0$
$x_0<0$	$-a^{1/p}$	$a^{1/p}$*	$-(-a)^{1/p}$
$x_0>0$	$a^{1/p}$	$a^{1/p}$	$-(-a)^{1/p}$*

* În acest caz nici un termen x_n nu trebuie să fie nul.

Evident : $x_0 = x_1 \Rightarrow x_0 = x_1 = \ldots = x_n = \ldots$

x_0 trebuie să fie de asemenea dat.

Eroarea absolută, s-o notăm ε, servește la oprirea calculului. Metoda lui Newton ne dă posibilitatea să evaluăm eroarea $|x_n - \sqrt[p]{a}|$ majorînd-o astfel :

$$|x_n - \sqrt[p]{a}| < |x_1| \prod_{i=1}^{n-1} \left[\left(\frac{p-1}{p}\right)\left(1 - \frac{a}{(x_i)^p}\right)\right].$$

Deci :

$$|x_1| \prod_{i=1}^{n-1} \left[\left(\frac{p-1}{p}\right)\left(1 - \frac{a}{(x_i)^p}\right)\right] < \varepsilon \Rightarrow |x_n - \sqrt[p]{a}| < \varepsilon.$$

11. a) Să se scrie algoritmul în LPS care extrage dintr-un șir de valori reale dat valoarea cea mai mare și o înscrie pe banda de ieșire. Șirul este dat pe banda de intrare precedat de o valoare întreagă pozitivă reprezentînd numărul de valori reale din șir (mai mic decît 1 000).

b) Încercați să rezolvați această problemă folosind un număr cît mai mic de celule ale memoriei.

Indicație. La un moment dat nu sînt necesare în memorie toate valorile reale.

12. Se dau două puncte într-un spațiu tridimensional prin coordonatele lor (valori reale). Să se scrie în LPS algoritmii ce rezolvă următoarele probleme :

a) Să se afle distanța dintre cele două puncte ;

b) Știind că cele două puncte sînt vîrfurile a doi vectori cu originea în centrul axelor de coordonate, să se afle produsul scalar al celor doi vectori, unghiul dintre cei doi vectori și coordonatele vîrfului produsului vectorial al celor doi vectori ;

c) Să se generalizeze problemele a) și b) pentru un spațiu n-dimensional (cu n dat).

13. Se dă banda de intrare din problema 11 (un șir de valori reale precedate de numărul lor). Să se scrie algoritmul în LPS care înscrie pe banda de ieșire :

a) suma valorilor din șir,

b) produsul valorilor din șir,

c) produsul valorilor nenule din șir,

d) suma pătratelor valorilor din șir,

e) media aritmetică a valorilor din șir,

f) media geometrică a valorilor pozitive din șir,

g) media armonică a valorilor nenule din șir,

h) dispersia valorilor din șir și abaterea medie pătratică.

Indicație. Dacă $x_1, x_2, \ldots, x_n$ sînt valori din șir, dispersia D și abaterea medie pătratică σ sînt date de relațiile :

$$D = \frac{\sum_{i=1}^{n} (x_i - m)^2}{n}, \quad \sigma = \sqrt{D}, \quad \text{unde } m = \frac{\sum_{i=1}^{n} x_i}{n} \text{ este media aritmetică a valorilor.}$$

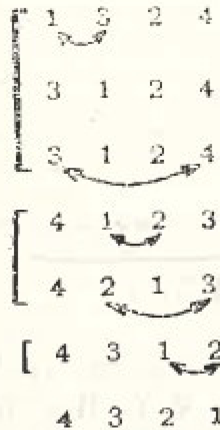
14*. *Variante ale algoritmului de ordonare.*

a) Analizați algoritmul de ordonare din figura I.27 bazat pe metoda „bulei" sau a „propagării" și observați că după a j-a parcurgere a șirului de numere, ultimele j valori sînt deja ordonate. Deci limita superioară pentru creșterea lui i poate fi $n - (j + 1)$ în loc de $n - 1$ (unde j numără parcurgerile șirului).
Rescrieți algoritmul care să includă această optimizare.

b) Mai mult, datorită unei eventuale ordonări parțiale a șirului încă de la începutul execuției s-ar putea ca mai mult de j valori de la sfîrșitul șirului să fie ordonate : dar cum putem ști cîte sînt ? O metodă simplă constă în memorarea poziției din șir la care s-a făcut ultima permutare a valorilor. Într-adevăr, dacă la o parcurgere a șirului ultima permutare s-a făcut la al k-lea element, atunci acesta și următoarele $n - k$ elemente sînt ordonate (dovediți acest lucru !). La următoarea prcurgere este deci suficient să căutăm perechile rău ordonate doar pînă la al $(k - 1)$-lea element. Rescrieți algoritmul de ordonare descrescătoare care să includă și această optimizare.

c) Prezentăm în cele ce urmează o altă metodă de ordonare. Ea se bazează pe parcurgerea șirului de numere pentru fiecare poziție și plasarea la sfîrșitul parcurgerii în poziția respectivă a valorii definitive. În acest fel, la a j-a parcurgere, primele $j - 1$ poziții sînt bine ordonate, deci valoarea din celula a j-a se compară doar cu valorile din celulele de la $j + 1$ la n. De fiecare dată cînd se găsește că a j-a valoare este mai mică

decît cea cu care se compară, cele două valori se permută.
Deci pentru un șir de 4 numere vom avea următoarele momente mai importante în ordonare:



Să se scrie algoritmul în LPS bazat pe această metodă.

d) Fie algoritmul de ordonare descrescătoare descris în figura I.32 printr-o schemă logică. Metoda folosită se numește metoda „inserării directe". Să se analizeze metoda și să se scrie algoritmul în LPS bazat pe această metodă.

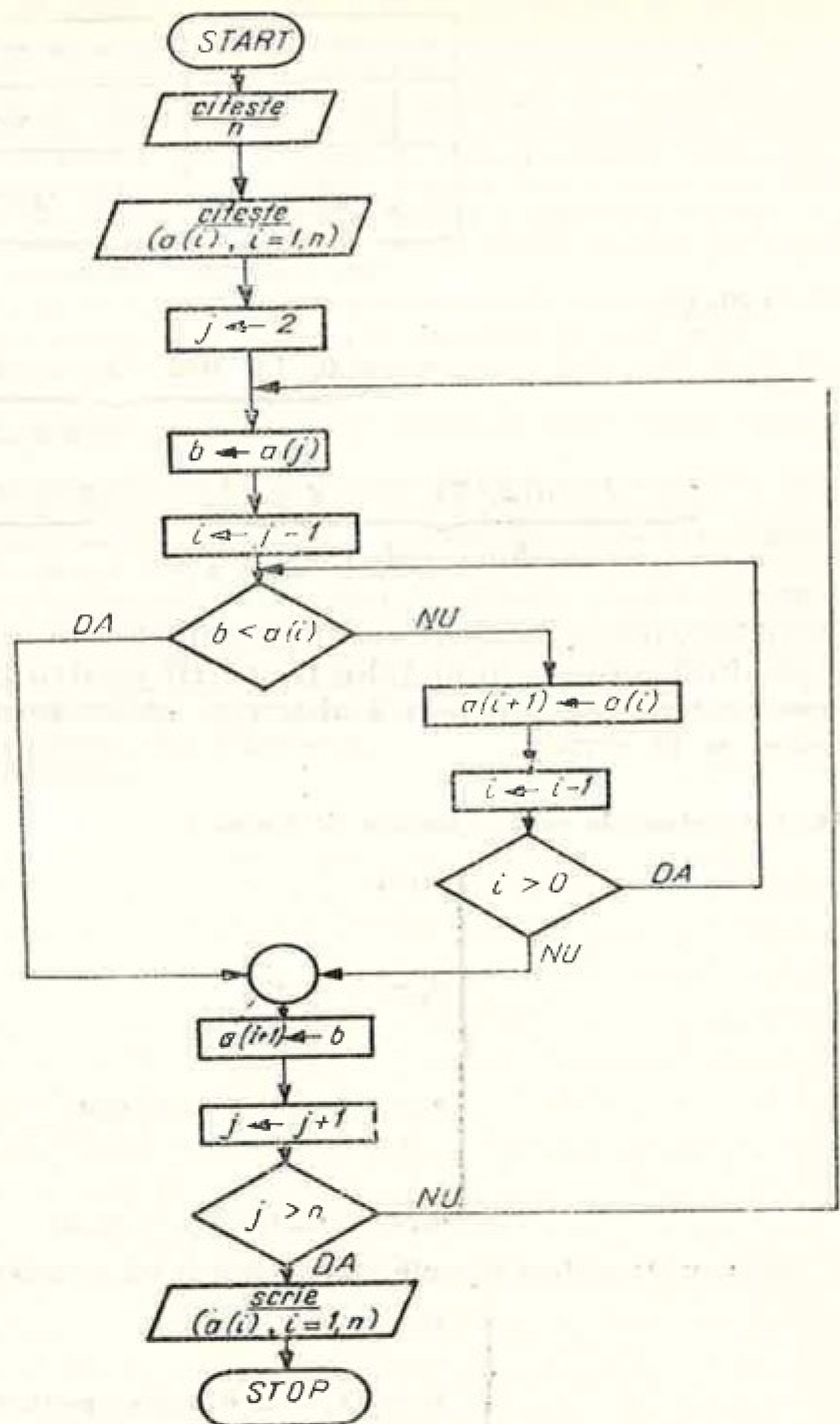


Fig. I.32

15. Să se scrie algoritmul în LPS pentru adunarea a două matrice de valori întregi. În afara valorilor (date în ordinea liniilor), pe banda d intrare se dau două numere întregi pozitive reprezentînd numărul liniilor, respectiv al coloanei matricei.

Instrucțiunile de citire și scriere ale tablourilor cu două dimensiuni sînt extinderi ale notației folosite pentru tablourile cu o dimensiune. Să presupunem citirea tabloului a declarat ca în figura I.33. Vom nota cu:

— $a(i, j)$ termenul general al tabloului a,

— $(a(i, j), i = 1, n)$ o listă de elemente ale tabloului a aflate pe coloana j adică: $a(1, j)$, $a(2, j)$, ..., $a(n, j)$.

$((a(i, j), i = 1, n), j = 1,m)$ ne spune să repetăm de m ori lista $(a(i, j), i = 1, n)$ înlocuind pe j cu 1, 2, ..., m.

n	var. simplă	valori întregi [1, 20]
m	var. simplă	valori întregi [1, 15]
a	tablou 2 dimens. (20, 15)	valori reale

Fig. I.33

Adică :

$$\underbrace{a(1,\ 1),\ a(2,\ 1), \ldots, a(n,\ 1),}_{\text{prima coloană}}$$

$$\underbrace{a(1,\ 2),\ a(2,\ 2), \ldots, a(n,\ 2)}_{\text{a doua coloană}}, \ldots, \underbrace{a(1,\ m),\ a(2,\ m) \ldots, a(n,\ m).}_{\text{a } m\text{-a coloană}}$$

Să observăm că la citire ordinea celulelor în care se citesc valorile trebuie să reproducă ordinea numerelor de pe BI pentru ca încărcarea valorilor în celulele matrice să reproducă intenția utilizatorului. Aceleași notații le vom folosi și la scriere.

16. Un sistem de ecuații liniare de forma :

$$\begin{cases} a_{11}x_1 & = b_1, \\ \vdots & \\ a_{21}x_1 + a_{22}x_2 & = b_2, \\ \vdots & \\ a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{ii}x_i & = b_i, \\ \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n & = b_n \end{cases}$$

se numește sistem triunghiular. Soluțiile lui se calculează pe rînd astfel :

$$\begin{cases} x_1 = b_1/a_{11} \\ x_i = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j)/a_{ii} \text{ pentru } i = 2, \ldots, n. \end{cases}$$

Să se scrie algoritmul în LPS care, primind la intrare numărul ecuațiilor și al variabilelor (cel mult 20), matricea coeficienților și vectorul termenilor liberi, tipărește la ieșire soluțiile sistemului.[1]

17. Se dă pe banda de intrare o matrice pătrată de valori reale. Să se scrie algoritmul în LPS care înscrie pe banda de ieșire suma elementelor aflate în triunghiul superior al matricei mărginit de cele două diagonale.[1]

18. Se dă un polinom prin gradul (valoare întreagă pozitivă) și coeficienții săi (valori reale).

a) Să se scrie algoritmul în LPS care calculează valoarea polinomului într-un punct dat (valoarea reală).

b) Să se scrie algoritmul în LPS care calculează valoarea derivatei polinomului într-un punct dat (valoare reală).

---

[1] Vezi indicația de reprezentare a tablourilor cu două dimensiuni de la problema 15.

19*. Se dă pe banda de intrare un şir de valori reale precedat de numărul acestora (mai mic decît 1 000). Să se scrie un algoritm în LPS care să tipărească pe banda de ieşire toate valorile distincte din şir urmate de numărul lor de apariţii în şir.

Indicaţie. Se va folosi un tablou cu două coloane.[1] În prima coloană se memorează numerele distincte, în a doua coloană numărul lor de apariţii. Pentru fiecare număr citit de pe banda de intrare, el este căutat printre numerele primei coloane. Dacă este găsit creşte cu 1 numărul de apariţii; dacă nu este găsit se înscriu într-o nouă linie din tablou numărul citit şi 1 ca număr de apariţii. La sfîrşitul citirii se tipăreşte tabloul.

20. Să se scrie algoritmul în LPS care să determine dacă un număr întreg pozitiv dat este număr prim sau nu. Ieşirea va fi un mesaj: „da“ sau „nu“.

Indicaţie. Se împarte numărul, fie el n, la 2 şi la toate numerele naturale de la 3 la $[n/2]$ sau chiar $[\sqrt{n}]$. Dacă măcar un rest este egal cu 0, numărul nu este prim.

21. Să se scrie algoritmul în LPS de generare a tuturor numerelor prime mai mici decît un număr întreg pozitiv dat.

22. Se scrie algoritmul în LPS care tipăreşte toţi divizorii primi ai unui număr întreg pozitiv dat.

23. *Problema maimuţei.* Pe o insulă pustie se află trei marinari naufragiaţi şi o maimuţă. Marinarii au reuşit să strîngă o grămadă de nuci de cocos şi s-au hotărît să le împartă frăţeşte a doua zi. Peste noapte, unul din marinari s-a sculat, a împărţit grămada în trei părţi egale după care rezultînd o nucă în plus a dat-o maimuţei, iar el a ascuns una din părţi. Pînă dimineaţa povestea s-a repetat şi cu ceilalţi doi marinari. Dimineaţa, marinarii împart grămada rămasă în trei părţi şi din nou rămîne o nucă de cocos pe care o dau maimuţei. Cîte nuci de cocos au fost la început în grămadă? Deoarece rezultatul nu este unic, să se găsească toate valorile mai mici de 1 000 care satisfac condiţia.

Observaţii. 1° Problema reprezintă un caz tipic pentru rezolvare cu ajutorul calculatorului, necesitînd calcule relativ simple, dar laborioase.

2° Prelucrarea nu are date de intrare.

---

[1] Vezi indicaţia de reprezentare a tablourilor cu două dimensiuni de la problema 15.

Capitolul II

PROGRAMAREA ÎN LIMBAJUL FORTRAN

II.1. DATE FORTRAN. INSTRUCŢIUNEA DE ATRIBUIRE. EXPRESII

II.1.1. Introducere

Asigurarea unei informări corecte, rapide şi utilizabile în conducerea unor activităţi umane constituie, în ultimii ani, un deziderat de cea mai mare importanţă. Ea se face simţită la toate nivelele societăţii, constituind unul dintre elementele determinante în obţinerea unor rezultate bune în activitatea desfăşurată. Conducerea activităţii unei uzine, stabilirea, planificarea, lansarea şi urmărirea operaţiilor de realizare a unui obiectiv (bloc de locuinţe, fabrică, instalaţie industrială etc.), supravegherea unor procese complexe (lansarea şi urmărirea evoluţiei unei rachete, comanda maşinilor-unelte sau a laminoarelor) constituie doar cîteva din domeniile unde obţinerea informaţiilor prin prelucrarea datelor este de cea mai mare importanţă. Unul din mijloacele cele mai perfecţionate care-i oferă omului posibilitatea prelucrării unui volum mare de date, în timp util, în scopul obţinerii informaţiilor necesare este calculatorul electronic.

Existenţa fizică a calculatorului electronic, fie el oricît de perfecţionat, nu răspunde necesităţilor sus-amintite. Este necesar să dispunem, de asemenea, de mijloace de comunicare adecvate între om şi calculator — limbajele — precum şi de oameni care să poată folosi noile posibilităţi de prelucrare a datelor pentru soluţionarea problemelor concrete — programatorii.

Limbajul FORTRAN constituie unul dintre mijloacele de descriere a prelucrărilor ce trebuie realizate de calculatorul electronic. Numele său provine din cuvintele englezeşti FORmula TRANslation care au semnificaţia de „traducerea formulelor". Creat în 1956 pentru calculatoarele din prima generaţie acesta este şi astăzi cel mai folosit limbaj. Aria sa de aplicabilitate s-a extins şi la probleme care necesită în primul rînd alte prelucrări decît calculele, aşa-numitele probleme economice. Acest limbaj poate fi învăţat destul

de rapid, avînd o sintaxă și o semantică simple. Programele scrise în FORTRAN sînt traduse cu ajutorul unor compilatoare adecvate, codul rezultat fiind apropiat de codul mașină și el poate fi executat de către calculatorul real. Alte avantaje ale limbajului FORTRAN sînt alcătuirea ușoară a programelor, depanarea rapidă, precum și asemănarea dintre descrierea calculelor în FORTRAN și în practica curentă.

Pentru a asigura compatibilitatea programelor, limbajul FORTRAN a fost standardizat. Aceasta înseamnă că sintaxa și semantica limbajului au fost fixate și că orice compilator construit pentru un calculator trebuie să le respecte. Astfel, dacă pe calculatoare diferite există compilatoare care respectă standardul FORTRAN, programele pot fi rulate pe oricare dintre acestea, obținîndu-se aceleași rezultate. De asemenea, a fost posibilă construirea unor colecții de programe care oferă soluțiile unor probleme matematice des întîlnite în aplicații (rezolvări de ecuații sau de sisteme algebrice, calcule cu matrice, calcule statistice etc.). Aceste colecții — denumite *biblioteci de programe* — pot fi folosite ușor de către toți programatorii interesați, reducîndu-se astfel timpul și efortul necesare rezolvării problemelor.

Un alt avantaj al limbajului FORTRAN (ca, de altfel, al tuturor limbajelor de nivel înalt) constă în faptul că în acesta se programează mai rapid decît în limbajul de asamblare al calculatorului. Mai mult, cel ce programează nu are nevoie să cunoască limbajul de asamblare, iar despre structura calculatorului trebuie să aibă doar cunoștințe generale.

II.1.2. Etapele rezolvării unei probleme utilizînd limbajul FORTRAN

Descrierea prelucrărilor care vor fi efectuate în limbajul FORTRAN — *scrierea programului* — constituie prima etapă în rezolvarea unei probleme. Deși deosebit de LPS, vom regăsi și în FORTRAN elementele de bază ale acestuia. Totuși aceste elemente sînt specifice limbajului FORTRAN, lucru normal dacă ținem seama că programele scrise vor fi rulate pe un calculator real, și nu pe un calculator fictiv.

Scopul alcătuirii programului FORTRAN — obținerea soluției unei probleme concrete — va fi atins în momentul *execuției programului* folosind datele necesare. Aceasta poate fi considerată ultima etapă în rezolvarea unei probleme folosind calculatorul (vezi figura II.1). Totuși între scrierea programului și obținerea rezultatelor există o etapă destul de laborioasă în care se face *punerea la punct a programului*. În această etapă programatorul va căuta să verifice dacă programul său îndeplinește sau nu prelucrarea dorită. Astfel, dacă programul său va fi destinat rezolvării unor ecuații de gradul al III-lea, în această etapă se va verifica dacă el rezolvă corect anumite ecuații ale căror soluții sînt cunoscute.

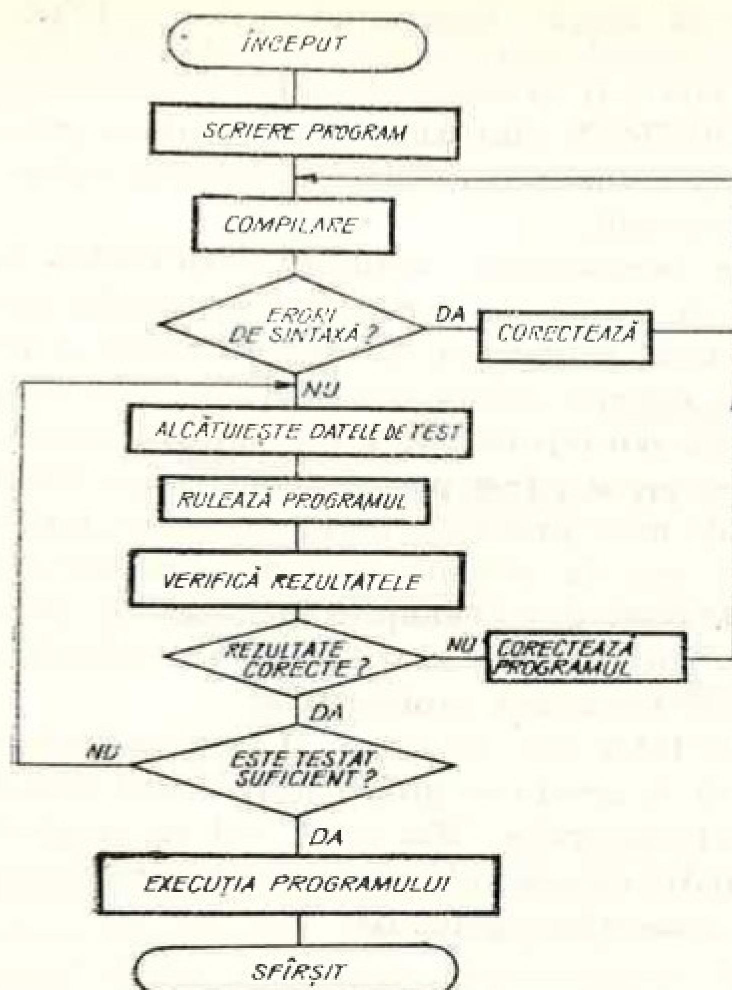


Fig. II.1

În etapa de punere la punct a programului, programatorul desfășoară o serie de activități, cu și fără ajutorul calculatorului. Dintre activitățile pentru care este nevoie de calculator, amintim compilarea programului și rularea programului cu date de test. Între activitățile descrise mai sus, programatorul corectează programul (înlătură erorile de sintaxă, schimbă algoritmul dacă este cazul etc.), depanează programul (identifică cauza erorii și o înlătură), alcătuiește datele de test, verifică rezultatele obținute ca urmare a rulării de test. Această etapă, punerea la punct a programului, este cea mai lungă în cadrul ciclului de elaborare a programului. De regulă se consumă pentru ea între 30% și 90% din timpul total al realizării programului care rezolvă problema respectivă.

Scrierea programului. Instrucțiuni. Formularul FORTRAN

FORTRAN fiind un limbaj de programare are reguli de scriere a programelor bine precizate și orice abatere de la ele conduce la semnalarea unor erori în momentul rulării pe calculator.

● Un program FORTRAN este alcătuit dintr-un șir de instrucțiuni. Instrucțiunile limbajului sînt formate din cuvinte care, la rîndul lor, sînt compuse din caractere. În FORTRAN pot fi folosite doar următoarele caractere[1] :

— literele mari ale alfabetului, ABCDEFGHIJKLMNOPQRSTUVWXYZ

— cifrele : 0123456789

— semnele speciale : + — = / ( )' · , și spațiu

● Instrucțiunile limbajului sînt de două feluri :

— *executabile*, care cer îndeplinirea unei acțiuni efective, din care va fi generat programul în limbaj mașină (cînd nu se poate naște o confuzie le vom numi pe scurt instrucțiuni) ;

[1] Celelalte caractere care pot fi afișate de către un calculator particular (de exemplu : &, ≠ etc.) pot fi utilizate numai în date de tip șir de caractere. Spațiul îl vom nota în continuare cu ␣.

— *neexecutabile* (de obicei acestea vor fi numite *declarații*), care descriu caracteristicile datelor utilizate, clasificarea programelor, indicații asupra formei datelor din seturile de date folosite etc.

● Structura unui program FORTRAN este de regulă următoarea : în prima parte declarațiile, apoi instrucțiunile care descriu prelucrările efective ce vor fi realizate asupra datelor descrise în partea de declarații, iar ultima instrucțiune din program trebuie să fie întotdeauna END.

Reamintim că *programul FORTRAN este descrierea unor prelucrări*. Instrucțiunile executabile descriu, în general, un pas sau o anumită parte a algoritmului care stă la baza prelucrării. Această parte a programului descrie prelucrarea propriu-zisă. Modul în care, pe baza instrucțiunilor, se îndeplinește prelucrarea dorită este următorul :

— se stabilesc atributele datelor utilizate, conform declarațiilor ;

— se realizează acțiunea cerută de prima instrucțiune executabilă ;

— dacă instrucțiunea curentă (în curs de execuție) nu indică executarea obligatorie a unei alte instrucțiuni se va îndeplini în continuare acțiunea cerută de instrucțiunea care urmează în program.

● În FORTRAN *forma generală a unei instrucțiuni* este următoarea :

<etichetă> || *<instrucțiune>*

<etichetă> este un număr întreg cuprins între 1 și 99999 și are drept scop identificarea unei instrucțiuni în cadrul programului. Acest număr îl vom numi etichetă. Trecerea la execuția altei instrucțiuni decît cea care o urmează pe aceea în curs de execuție se realizează cerînd în program *să se execute obligatoriu instrucțiunea cu eticheta respectivă*.

Eticheta marchează deci o anumită instrucțiune a programului, atașîndu-i un număr, pe baza căruia poate fi identificată. Trebuie precizat că nu este necesar ca orice instrucțiune să aibă etichetă. În același timp, două instrucțiuni nu pot avea aceeași etichetă, lucru evident datorită scopului folosirii acestora — identificarea instrucțiunii. În figura II.2 este dat un exemplu de program FORTRAN, care rezolvă o ecuație de gradul I ai cărei coeficienți sînt citiți de pe o cartelă.

● Instrucțiunile, care în FORTRAN au un format prestabilit, se pot scrie pe liniile unui formular de programare standard, împărțit în 80 de coloane (vezi fig. II.2). Pe fiecare linie a formularului se poate scrie o instrucțiune. Scrierea este liniară, neadmițîndu-se notațiile sub linie sau deasupra ei ca A_1 sau B^4, iar caracterele ce compun instrucțiunile trebuie scrise cîte unul în fiecare coloană.

Pachetul de cartele rezultat din perforarea instrucțiunilor de pe formular va alcătui *programul sursă* ce va fi apoi citit și compilat de calculator.

O linie a formularului este compusă din patru zone astfel :

1. Primele 5 coloane alcătuiesc *zona pentru etichete*.

FORTRAN Formular de programare

Program: Rezolvarea ecuației de gradul I		INDICAȚII PENTRU PERFORARE					Fila 1 din 1
		Semne	(	)	=	+	
Programator: Preoteasa Petre	Data: 17.II.80	Perforare					

```
Eticheta |C| INSTRUCȚIUNI FORTRAN                              | Observații
1 Nr   5 |6|7  10   15   20   25   30   35   40   45  ...  72 | 73  75  80
C   REZOLVAREA  ECUATIEI DE GRADUL I                          |        10
          REAL A,B,X                                          |        20
          READ(105,11) A,B                                    |        30
          IF(A.EQ.0) GØTØ 1                                   |        40
              X = -B/A                                        |        50
              WRITE(108,12) X                                 |        60
              GØTØ 3                                          |        70
     1    IF(B.EQ.0) GØ TØ 2                                  |        80
              WRITE(108,13)                                   |        90
              GØTØ 3                                          |       100
     2    WRITE(108,14)                                       |       110
     3    STØP                                                |       120
    11    FØRMAT(2F6.2)                                       |       130
    12    FØRMAT('1','SØLUTIE: X=',F6.2)                      |       140
    13    FØRMAT('1','NICI Ø SØLUTIE')                        |       150
    14    FØRMAT('1','Ø INFINITATE DE SØLUTII')               |       160
          END                                                 |       170
```

Fig. II.2

	Eticheta 1	2	3	4	5	Cont. 6	7 8 9 10 11 12 13 14 15 16 17 18
			1	5	6		
sau		1		5	8		
sau	1	5	8				
sau	1		5		8		
sau	1	5	8				
sau	1		5		8		

Fig. II.3

Blancurile ce apar între cifre sînt ignorate, după cum se vede în figura II.3. Cifrele unei etichete nu pot fi scrise în afara acestei zone.

2. Coloana a 6-a — de *continuare.* Dacă o instrucțiune nu încape pe o singură linie de formular, scrierea ei se poate continua pe liniile următoare punînd un caracter diferit de zero sau blanc, în coloana a 6-a. Sînt admise cel mult 19 linii de continuare pentru o instrucțiune. Zonele de etichetă corespunzătoare liniilor de continuare trebuie să fie libere. Un exemplu este dat în figura II.4.

coloană de continuare

Eticheta 1 2 3 4 5	Cont. 6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 … 72	73 74 75 76 77 78 79 80
1 2 5		X 1 = A L F A + (B E T A *	R 4 5 / 1 2 1 0
	1	G A M A / 1 2 5 . 7 3) +	R 4 5 / 1 2 2 0
	2	(R E Z + Y) / 2 5 . 7	R 4 5 / 1 2 3 0

Fig. II.4

3. În *zona de instrucțiuni* (între coloanele a 7-a și a 72-a) se descriu instrucțiunile. Blancurile care apar între cuvinte sînt ignorate de compilator, ele fiind folosite numai pentru a ușura citirea instrucțiunii.

Exemplu : Instrucțiunile următoare sînt echivalente :

|| READ (105, 25) A, B, C

|| READ ␣␣(105, 25)␣␣␣A, ␣␣ B␣, ␣␣C

4. *Zona de identificare* (coloanele 73—80) servește pentru numerotarea rîndurilor sau pentru scurte comentarii. Datorită faptului că e posibilă o amestecare a caracterelor programului sursă, este utilă o numerotare a lor. Informațiile conținute în această zonă nu sînt luate în considerare de către compilator, ele fiind doar listate pe imprimantă.

Comentarii. Pentru a face programul cît mai explicit se pot adăuga informații suplimentare privind notații sau etape ale procesului de calcul, cu ajutorul unor linii de comentariu. O linie de comentariu se deosebește de celelalte, prin faptul că are un „C" în coloana 1. Comentariile pot fi scrise apoi, din coloana 2 pînă în 80.

Liniile de comentariu pot apărea oriunde în program. Ele nu influențează prelucrarea, fiind doar listate la imprimantă împreună cu programul.

Rularea programelor scrise în limbajul FORTRAN

După ce programul sursă a fost perforat pe cartele este nevoie ca el să fie compilat, corectat, testat etc. Aceste activități se desfășoară, după cum am mai spus, cu ajutorul calculatorului.

Pentru a rula un program pe un calculator din seria FELIX-C este nevoie ca programatorul să alcătuiască o lucrare în care, în afara programului sursă, se mai adaugă și cartelele de comandă care dau informații sistemului de operare asupra modului cum se dorește să fie executată lucrarea. Vom da în continuare cartelele de comandă pentru rularea unui program FORTRAN care citește datele de pe cititorul de cartele și scrie rezultatele pe imprimantă :

```
↓col. 1      col. 11
  •          JOB nume1, AN : cont. PN : nume 2
  •          COMPILE FORTRAN
 ┌─────────────────────────────────┐
 │           program sursă         │
 └─────────────────────────────────┘
  •          LINK
  •          RUN
 ┌─────────────────────────────────┐
 │           date                  │
 └─────────────────────────────────┘
  . EOF
  •              EOJ
```

Data *cont* va fi dată de centrul de calcul pe al cărui calculator se va face rularea. Datele *nume* 1 și *nume* 2 sînt fixate de programator reprezentînd numele lucrării și respectiv al programatorului.

II.1.3. Date FORTRAN

Programul descrie de fapt modalitatea de transformare a datelor de intrare în date de ieșire. Calculatorul va îndeplini, în momentul execuției, acțiunile cerute de instrucțiunile programului. Fiind un limbaj destinat descrierii prelucrărilor care implică calcule numeroase, în FORTRAN sînt create facilitățile necesare pentru definirea și utilizarea unei mari varietăți de date.

În FORTRAN datele sînt grupate în două categorii :

— *constante* ale căror proprietăți și valori sînt deduse din forma lor de scriere ;

— *variabile* ale căror proprietăți sînt descrise într-o declarație iar valorile le sînt atribuite prin instrucțiuni executabile.

Trebuie precizat că, dacă o constantă are o singură valoare în tot timpul execuției programului, în schimb o variabilă poate să-și schimbe valoarea de mai multe ori. De asemenea, trebuie reamintit că o variabilă nu poate avea două valori în același timp.

Atributele datelor **FORTRAN** sînt *tipul* si *structura*. Tipurile permise pentru date sînt : *întreg, real, precizie dublă, complex, logic* şi *caracter* (denumit şi literal sau Hollerith). Structurile admise sînt : *variabilele simple* şi *tablourile*.

Pentru a înţelege diferenţa dintre tipurile de date, precum şi necesitatea folosirii lor, trebuie să pornim de la reprezentările externe ale datelor folosite în mod obişnuit de către om. Fiecărui tip de dată în formă externă îi corespunde o reprezentare, într-o formă internă, proprie calculatorului.

În general utilizatorul nu cunoaşte forma internă de reprezentare a datelor. Dar el trebuie să folosească formele externe de reprezentare a datelor conform restricţiilor impuse de limbaj.

Forma internă de reprezentare a numerelor este diferită de la un calculator la altul. Compilatorului i se cere să stabilească pentru fiecare dată FORTRAN forma internă de reprezentare a valorii ei. Pentru a face acest lucru trebuie ca următoarea regulă generală să fie îndeplinită la descrierea datelor.

Într-un program, fiecare dată folosită trebuie să fie de tip permis în FORTRAN. O dată nu poate fi de două tipuri diferite în acelaşi program.

În FORTRAN pot fi folosite constante de oricare din tipurile admise. În cazul acestora, forma externă apare în program şi pe baza ei compilatorul va deduce tipul şi valoarea constantei. Pentru variabile, valoarea poate fi afişată sau citită sub forma externă. În program se utilizează numai numele variabilei. *Numele unei variabile poate fi orice şir de litere sau cifre care începe cu o literă*, compus din cel mult 8 caractere. Cîteva exemple pot fi : A25, ALFA, B, C2345, C2346, C234X34.

Tipul variabilelor poate fi întreg, real, precizie dublă, complex şi logic. Nu există variabile de tip caracter.

Tablourile desemnează grupări de variabile de acelaşi tip. Fiecare variabilă din tablou poate fi referită prin numele tabloului urmat, între paranteze rotunde, de indici, care arată poziţia ei în cadrul grupului. Numele de tablouri se alcătuiesc după aceleaşi reguli ca şi cele de variabile. Astfel, dacă tabloul A are 2 linii şi 3 coloane, elementele lui vor fi : A(1,1), A(1,2), A(1,3), A(2,1), A(2,2), A(2,3). Pentru a deosebi variabilele care sînt elemente ale unui tablou de variabile simple, le vom numi *variabile indexate*. Trebuie menţionat că şi în cazul tablourilor nu există posibilitatea de a folosi variabile indexate de tip caracter.

În continuare dăm pentru fiecare tip de date permis, principalele caracteristici.

Tipul întreg. *Forma externă.* Număr întreg cu sau fără semn. În reprezentare nu poate apărea punctul zecimal[1]. Semnul + este subînţeles dacă întregul nu are semn. *Exemple*: 111 —2175 0 4598345. *Valoarea v* asociată formei externe este aceea a numărului zecimal reprezentat. *Ordinul de mărime* $|v| \leqslant 2^{31} - 1$ sau $|v| \leqslant 2147483647$. *Forma internă.* Reprezentarea exactă a valorii numărului, în virgulă fixă.

Tipul real. *Forma externă.* Numărul real în reprezentarea căruia apare de regulă punctul zecimal sau un exponent. Această reprezentare se face în două forme :

---

1 În limbajul FORTRAN cînd se foloseşte punctul zecimal pentru reprezentarea unei valori în forma externă, acesta are rol identic cu virgula zecimală aşa cum se cunoaşte.

Forma I. Şir de cifre conţinînd un punct zecimal ce separă partea întreagă de partea fracţionară. Şirul poate fi precedat de semnul + sau —. Partea fracţionară şi partea întreagă nu pot să lipsească în acelaşi timp. *Exemple :* +45.567 —123.896 .789 —6781. 0.0

Valoarea v asociată este aceea a numărului zecimal cu aceeaşi scriere.

Forma II. Este similară cu scrierea numerelor zecimale sub forma : $\pm a_1 a_2 \ldots a_p \cdot a_{p+1} \ldots a_n \times 10^q$. În FORTRAN în locul notaţiei 10^q se va folosi notaţia Eq unde q este exponentul. Deci forma generală de reprezentare este : *<fracţie>* E± *<exponent>* unde *<fracţie>* poate fi reprezentată ca o dată reală în forma I sau ca una de tip întreg. *Exemple :* 31.458E—2 —234E23 +.1985E+ 16 2731849E—6

Valoarea reprezentată va fi : *<fracţie>* × 10 *<exponent>* *Ordinul de mărime* este $10^{-77} < |v| < 10^{77}$. Reprezentarea este aproximativă reţinîndu-se cel mult 7 cifre semnificative. Exponentul poate avea cel mult două cifre. *Forma internă* pentru cele două moduri de reprezentare este reprezentarea aproximativă a valorii numărului în virgulă mobilă format scurt (de regulă pe 4 baiţi).

Tipul precizie dublă. *Forma externă.* Numărul real în reprezentarea descrisă la tipul real, cu următoarele deosebiri : în locul literei E se foloseşte litera D pentru *<exponent>*, iar pentru valoare se pot păstra pînă la 16 cifre semnificative. Restul caracteristicilor coincid cu cele ale datelor de tip real.

Tipul complex. *Forma externă.* O pereche de numere reale, care reprezintă partea reală şi, respectiv, partea imaginară. *Exemple :* (21.0,1.0) (2.7, —82.756) (.12345, —179E —2).

Trebuie precizat că aceasta este reprezentarea constantelor de tip complex. În cazul valorii variabilelor de tip complex, parantezele nu mai sînt folosite, forma externă constînd din două date de tip real. *Ordinul de mărime, forma internă, valoarea părţii reale* şi a *părţii imaginare* se deduc uşor din faptul că se utilizează reprezentări de tip real.

Tipul logic. *Forma externă.* Valorile logice ADEVĂR şi FALS sînt reprezentate în instrucţiunile FORTRAN sub forma .TRUE. şi respectiv .FALSE. Imprimanta tipăreşte T pentru .TRUE. şi F pentru .FALSE.

Forma internă. O zonă de memorie (de regulă un cuvînt), al cărui conţinut permite identificarea celor două valori logice.

Tipul caracter (literal sau Hollerith). *Forma externă.* O dată literală este reprezentarea unei informaţii literale (şir de caractere) pentru procesul de calcul.

În instrucţiunile FORTRAN şirul de caractere este scos în evidenţă prin includerea lui între apostrofuri sau scriind înaintea acestuia *n*H unde *n* este numărul caracterelor din şir. La imprimare şirul de caractere este tipărit fără apostrofuri sau *nH*.

Se pot utiliza toate caracterele pentru care există o reprezentare în calculator. Dăm mai jos cîteva exemple de informaţii literale care pot fi reprezentate sub forma unei date literale :

Reprezentarea la ieşire	**Şirul de caractere** *Reprezentare în instrucţiuni*	
SCHIMBUL DOI	'SCHIMBUL DOI' sau	12HSCHIMBUL DOI
CLASA A XI	'CLASA A XI'	10HCLASA A XI

Forma internă. Fiecărui şir de caractere îi corespunde o zonă de memorie, conţinutul acesteia fiind stabilit în funcţie de data literală folosită.

II.1.4. DECLARAȚII FORTRAN

Constantele sînt folosite în programul sursă în forma externă pe care am prezentat-o anterior. Prin modul de scriere, constantele își definesc valoarea și tipul. Celorlalte date FORTRAN, trebuie să li se asocieze, într-un mod bine stabilit, un tip și o valoare. Modul cum unei date i se asociază o valoare (reamintim că o variabilă poate avea mai multe valori în timpul execuției unui program) va fi tratat în paragrafele următoare. Stabilirea unui tip asociat variabilei se face conform convenției predefinite, printr-o declarație explicită sau printr-o declarație implicită.

a) **Convenția predefinită** permite să se asocieze un tip variabilei, după următoarea regulă :

În cazul cînd nu se specifică nimic despre tipul unei variabile, dacă prima literă a numelui este I, J, K, L, M *sau* N, *tipul este întreg ; dacă este orice altă literă, tipul este real.*

Folosind convenția predefinită, programatorul poate alege *numele* variabilelor reale și întregi în așa fel încît să nu folosească declarații de tip în programul alcătuit.

Pentru a lucra cu variabile de celelalte tipuri sau a specifica un alt tip de variabilă decît cel rezultat din convenția predefinită se folosesc declarații de tip.

Aceste declarații pot fi explicite sau implicite.

b) **Declarația explicită** precizează tipul asociat unor variabile din programul respectiv.

Forma generală a declarației explicite este :

$$| \, | \; t \; v_1, \, v_2, \, \ldots, \, v_n$$

unde :

- t indică tipul și este unul din cuvintele : INTEGER, REAL, DØUBLE PRECISIØN, CØMPLEX sau LØGICAL.
- $v_1, v_2, \ldots, v_n$ este o listă de variabile care are cel puțin un element.

Semnificație. Variabilelor din listă li se asociază tipul t.

Lista reprezintă un șir de elemente separate prin virgulă. După ultimul element al listei nu trebuie pus nici un semn de punctuație.

Exemple :

Listă de numere :	41, —2, 3, 0 —2.20
Listă de variabile :	K1, K2, ALFA, MAX, Z
Listă de litere :	A, B, L, M, X, Y, Z

Trebuie reținut că declarațiile explicite nu pot avea etichete.

Exemple :

```
| | LØGICAL L1, L2, FALS
| | REAL MAX, INT, LUNG, R1, Z1, Z2
| | INTEGER ØRE, ZILE, ANI, NUL
| | CØMPLEX D, E
```

c) **Declarația implicită** este asemănătoare cu convenția predefinită, stabilirea tipului variabilei făcîndu-se după prima literă a numelui.

Forma generală :

$$| \, | \; \text{IMPLICIT} \; t_1(l_1), \; t_2(l_2), \ldots, t_n(l_n)$$

unde :

● $t_1, t_2, \ldots, t_n$ sînt cuvinte din mulțimea : INTEGER, REAL, DØUBLE PRECISIØN, LØGICAL și CØMPLEX.

● $l_1, \ldots, l_n$ liste ale căror elemente pot fi litere sau grupuri de două litere unite cu liniuță (prima literă din grup trebuie să o preceadă pe a doua în alfabet).

Semnificație. O variabilă al cărei nume începe cu o literă ce face parte din lista l_k $(1 \leqslant k \leqslant n)$ este de tipul t_k dacă numele ei n-a apărut într-o declarație explicită.

— Dacă elementul unei liste are forma $a-b$ unde a și b sînt litere ale alfabetului, atunci toate literele din alfabet cuprinse între a și b fac parte din lista respectivă.

Exemplu. Într-un program apare declarația :

```
| | IMPLICIT LØGICAL (L-R), CØMPLEX (A-D, X, Y), DØUBLE PRECISIØN
    (F, G)
```

Variabilele al căror nume încep cu literele :

— A, B, C, D, X sau Y sînt complexe,
— L, M, N, Ø, P, Q, R sînt logice,
— I, J, K sînt întregi.
— E, H, S, T, U, V, W, Z sînt reale,
— F, G sînt în precizie dublă,

Pentru stabilirea tipului asociat unei variabile se aplică următoarea regulă :

— *Dacă numele variabilei apare într-o declarație explicită atunci tipul ei este corespunzător acesteia,*

— *În caz contrar, tipul se stabilește după prima literă a numelui : dacă litera apare într-o declarație implicită atunci variabila are tipul stabilit de aceasta, altfel se aplică convenția predefinită.*

De asemenea trebuie reținut :

1. Declarațiile de tip dintr-un program nu trebuie să fie contradictorii.

Exemplu :

```
|| REAL MAX, L, L1, IDENT
|| CØMPLEX C1, GAMA, L3
|| LØGICAL L1, IDENT
|| . . . . . . . . . . . . . . . . . .
```

Conform acestor declarații variabile L1 *și* IDENT *ar trebui să aibă asociate cele două tipuri, lucru nepermis în* FORTRAN.

2. Declarațiile trebuie să apară la începutul programului, înainte de folosirea variabilelor. Dacă se folosește o declarație implicită ea trebuie să fie prima instrucțiune din program.

II.1.5. Instrucțiuni de atribuire

Limbajul FORTRAN este foarte potrivit pentru rezolvarea problemelor tehnice și științifice. În definirea oricărei probleme un rol important îl joacă formulele, care exprimă calculele ce trebuie făcute. În limbajul FORTRAN echivalentul unei formule este ***instrucțiunea de atribuire***. Forma generală a acesteia este :

$$|| <v> = <exp>$$

unde $<v>$ este o variabilă (simplă sau indexată) iar $<exp>$ este o expresie.

Semnificația instrucțiunii de atribuire este cea obișnuită : se atribuie variabilei $<v>$ valoarea rezultată din evaluarea expresiei $<exp>$. În exemplele următoare sînt prezentate cîteva instrucțiuni de atribuire :

A = 5.3 variabila A primește valoarea 5.3
B2 = 3.14*R/A variabila B2 primește valoarea 3.14*R/A

TAB(8) = A—B2 variabila TAB(8) primeşte valoarea expresiei A—B2
LOG = .TRUE. variabila logică LOG primeşte valoarea ADEVĂR.

Ca şi tipul variabilelor, valoarea expresiilor poate fi de unul din tipurile: întreg, real, precizie dublă, complex şi logic. Dacă tipurile lui <v> şi al expresiei *<exp>* coincid, atribuirea este imediată. În cazul în care ele sînt diferite, este făcută mai întîi o conversiune a valorii expresiei la tipul varia-bilei <v> şi abia după aceea este făcută atribuirea. Dacă această conversiune nu se poate face (vezi figura II.5) este evident că instrucţiunea de atribuire este incorectă.

v = exp		Tip expresie exp				
		I	R	PD	C	L
Tip variabilă v	I	A	CA	CA		
	R	CA	A	CA		
	PD	CA	CA	A		
	C				A	
	L					A

LEGENDĂ: I = întreg; R = real; C = complex; PD = precizie dublă; L = logic; A = atribuire imediată; CA = conversie şi atribuire

Fig. II.5

II.1.6. Expresii FORTRAN

Expresia FORTRAN *constituie reprezentarea în acest limbaj a unei expresii matematice obişnuite. Ea este o combinaţie a operatorilor, a operanzilor şi a parantezelor.* Pentru a descrie modul de alcătuire a unei expresii FORTRAN corecte, vom analiza mai întîi care sînt operatorii şi operanzii permişi.

a) *Operatorii* FORTRAN *care pot fi utilizaţi sînt de trei tipuri: aritmetici, relaţionali* şi *logici.*

OPERATORI

ARITMETICI		LOGICI		RELAŢIONALI	
FORTRAN	Matematică	FORTRAN	Matematică	FORTRAN	Matematică
+	+	.OR.	sau	.LT.	$<$
—	—	.AND.	şi	.GT.	$>$
*	×	.NOT.	negaţie	.EQ.	$=$
/	/			.LE.	$\leqslant$
**	ridicare la putere			.GE.	$\geqslant$
				.NE.	$\neq$

Tabloul funcţiilor FØRTRAN standard[1]

Funcţia	Definiţie	Numele funcţiei în FORTRAN	Nr. argumentelor	Tip: Argumente	Tip: Valoare funcţie
Valoare absolută	$\lvert a \rvert$	ABS IABS DABS CABS	1 1 1 1	R I D C	R I D R
Modul	$a_1 - \left[\frac{a_1}{a_2}\right] a_2$	MØD AMØD DMOD	2 2 2	I R D	I R D
Obţinerea părţii reale a unui argument complex		REAL	1	C	R
Obţinerea părţii imaginare a unui argument complex		AIMAG	1	C	R
Exprimă două argumente reale în formă complexă	$a_1 + a_2\sqrt{-1}$	CMPLX	2	R	C
Obţinerea conjugatului unui număr complex		CØNJG	1	C	C
Exponenţială	e^a	EXP DEXP CEXP	1 1 1	R D C	R D C
Logaritm natural	$\ln x$	ALØG DLØG CLØG	1 1 1	R D C	R D C
Logaritm zecimal	$\lg a$	ALØG 10 DLØG 10	1 1	R D	R D
Sinus	$\sin a$	SIN DSIN CSIN	1 1 1	R D C	R D C
Cosinus	$\cos a$	CØS DCØS CCØS	1 1 1	R D C	R D C
Arctangent	$\text{arctg } a$	ATAN DATAN	1 1	R D	R D
	$\text{arctg } (b/a)$	ATAN 2 DATAN 2	2 2	R D	R D
Rădăcina pătrată	$\sqrt{a}$	SQRT DSQRT CSQRT	1 1 1	R D C	R D C

[1] Notaţie: C — complex, I — întreg, R — real, D — precizie dublă.

Fig. II.6

b) Operanzi FORTRAN pot fi :

- Constante sau variabile (simple sau indexate)
- Anumite funcţii uzuale din matematică (rădăcină pătrată, funcţii trigonometrice, logaritmi etc.). Ele pot fi folosite în expresiile FORTRAN, în mod asemănător ca în matematică. Forma generală de descriere a utilizării unei asemenea funcţii (numită apel sau chemarea funcţie) este :

$$nume\text{-}funcţie\ (argument_1,\ argument_2,\ \ldots,\ argument_n)$$

dacă presupunem că funcţia are n argumente. Cînd o asemenea descriere apare într-o expresie FORTRAN înseamnă că valoarea funcţiei va fi calculată pentru argumentele indicate şi va fi folosită drept operand la evaluarea expresiei. În figura II.6 sînt date funcţiile matematice care se pot folosi în expresiile FORTRAN. Modul de alcătuire a apelului funcţiilor, ca şi tipul valorii reies din aceeaşi figură. Dăm mai jos cîteva exemple :

ABS(ALFA)	$\lvert\alpha\rvert$	ALOG10(B)	$\log_{10}b$
SQRT (6.25)	$\sqrt{6.25}$	ATAN (BETA)	arctg β

- Expresii FORTRAN închise între paranteze. Acest tip de operand îl vom numi *subexpresie*.

Exemple : (B*C) (—B + SQRT(B*B—4*A*C)) (2*A)

În concluzie, într-o expresie se pot folosi drept operanzi constante, variabile, funcţii sau subexpresii.

Fiecărui operand i se asociază un tip conform figurii II.7. Acest tip este important în evaluarea expresiei, după cum vom vedea.

c) *Evaluarea expresiilor*

Expresiile pot fi împărţite în două clase : aritmetice şi logice. Pentru expresiile aritmetice valoarea expresiei poate fi de tip întreg, real, precizie dublă sau complex. Tipul valorii expresiilor logice poate fi numai logic. Regulile după care se găseşte valoarea unei expresii le vom da pentru cele două clase, separat.

Operandul	Se asociază tipul
Constantă	Constantei
variabilă	variabilei
element tablou	tabloului
funcţie	funcţiei
subexpresie	evaluării subexpresiei

Fig. II.7

A ① B		Tipul operandului B			
		I	R	PD	C
Tipul operandului A	I	I	R	PD	C
	R	R	R	PD	C
	PD	PD	PD	PD	C
	C	C	C	C	C

A**B		Tip operand B			
		I	R	PD	C
Tip operand A	I	I	R	PD	C
	R	R	R	PD	
	PD	PD	PD	PD	
	C	C			

LEGENDA: ① una din operațiile + − * /   I = întreg   R = real
PD = precizie dublă   C = complex

Fig. II.8

Expresii aritmetice

Rezultatul unei operații aritmetice depinde de tipul operanzilor (aceștia nu pot fi de tip logic) așa cum rezultă din figura II.8 pentru adunare, scădere, înmulțire, împărțire și pentru ridicarea la putere.

Evaluarea expresiilor aritmetice este făcută după reguli precis stabilite. Vom da în primul rînd regulile după care sînt evaluate expresiile aritmetice fără paranteze (cele care nu conțin subexpresii).

S1. *Dacă în expresie sînt folosite funcții, valorile acestora sînt găsite și se substituie apelurilor respective.*

S2. *Operațiile aritmetice sînt efectuate conform următoarelor priorități :*

1) **

2) * și /

3) + și −

— *În caz de egalitate a priorității operațiilor, efectuarea operațiilor aritmetice se face de la stînga la dreapta, cu excepția ridicării la putere care se execută de la dreapta la stînga :*

A**B**C se evaluează cu A**(B**C)

S3. *Pentru efectuarea unei operații dacă operanzii nu sînt de același tip se convertesc valorile acestora în tipul rezultatului conform figurii II.8. Această regulă nu se aplică în cazul ridicării la o putere întreagă a unui număr de alt tip : exponentul acesteia va rămîne de tip întreg.*

Exemplul 1. Într-un program sînt folosite declarațiile :

```
IMPLICIT DØUBLE PRECISIØN (D, P, S, T), CØMPLEX (U−Z)
REAL G(50)
```

În descrierea modului cum sînt evaluate expresiile care urmează au fost folosite mai multe variabile intermediare, ale căror nume au fost astfel alese încît tipul lor să poată fi dedus din declarațiile de mai sus.

```
A/B + C*C**F    G(10) + B/C —    SIN(ALFA)**C + 1.567
A/B + C*R1      G(10) + B/C —    R1 **C       + 1.567
 R2 + C*R1      G(10) + B/C —    R2           + 1.567
  R2 + R3       G(10) + R3 —     R2           + 1.567
  Valoare           R4     —     R2           + 1.567
                           R5                 + 1.567
                                 Valoare
```

Evaluarea unei expresii aritmetice oarecare se face după regulile următoare :

R1. *Orice expresie aritmetică care nu conține subexpresii se evaluează după regulile S1, S2* și *S3.*

R2. *Subexpresiile conținute într-o expresie trebuie evaluate înainte de a interveni ca operanzi într-o operație aritmetică. Se începe cu evaluarea celei mai din interior subexpresii. După calculul valorii unei expresii se pune în locul ei valoarea găsită, continuîndu-se procedeul cu celelalte subexpresii. Cînd se ajunge ca expresia dată să nu mai conțină subexpresii, evaluarea se face după regula R1.*

Exemplul 2. Într-un program se află următoarele declarații :

```
IMPLICIT DØUBLE PRECISIØN (D, T—Z)
REAL F(25), Y
DØUBLE PRECISIØN X(100)
```

În cele ce urmează vom arăta cum se evaluează anumite expresii folosite în acest program. Variabilele ce vor reține valori intermediare ale calculelor vor fi de tipul acestor valori.

```
(A + B**(I/J)) * (((M/N — A/(B — SQRT(C)) — X(7))/F(5))
(A + B**I1)    * (((M/N — (A/(B — SQRT(C)) — X(7))/F(5))
      R1       * (((M/N) — A/(B — SQRT(C)) — X(7))/F(5))
      R1       * ((  I2 — A/(B — SQRT(C)) — X(7))/F(5))
      R1       * ((  I2 — A/(B —  R2)     — X(7))/F(5))
      R1       * ((  I2 — A/    R3        — X(7))/F(5))
      R1       * (              R4             /F(5))
Rezultă :  R1  *                R5
```

Expresii logice

În FORTRAN o expresie logică poate fi :

a) O constantă logică (.TRUE. sau .FALSE.).

b) O variabilă logică sau un element al unui tablou de tip logic.

c) Două expresii aritmetice unite printr-un operator de relaţie.
d) Expresii logice unite prin operatori logici.

Se pot alcătui expresii logice de forma : exp_1 R exp_2 unde exp_1, exp_2 sînt expresii aritmetice iar R e un operator de relaţie. Valoarea expresiei este găsită astfel : se evaluează expresiile aritmetice şi apoi se compară valorile lor în funcţie de operatorul de relaţie care apare între ele, rezultînd o valoare logică.

Exemplu. Variabilele A, B, M şi N au valorile 2.0, 1.5, 2 şi respectiv —3.

Dăm mai jos cîteva exemple de expresii logice şi valoarea lor :

A.NE.B	.TRUE.
A.EQ.(B + A)	.FALSE.
M.GT.(N*M)	.TRUE.
A.EQ.M	.TRUE.

Cînd expresiile aritmetice nu sînt de acelaşi tip, valorile vor fi convertite în tipul de rangul cel mai mare dat de următoarea ordine : întreg, real, precizie dublă, complex (în cazul exemplului ultim, în real).

Dacă expresiile logice sînt alcătuite folosind operatorii logici, evaluarea lor se realizează în funcţie de paranteze şi ţinînd seama de următoarele priorităţi : expresii aritmetice, expresii de relaţie, .NOT., .AND., .OR.

Exemple : Fie variabilele :

A, B, C, reale cu valorile 2.0, —1.5 şi respectiv —10.0
I, J întregi cu valorile 4 şi respectiv —2
L1, L2, L3 cu valorile .TRUE., .FALSE. şi respectiv .TRUE.

În acest caz următoarele expresii logice au valorile indicate :

A.GT.C.AND..NØT.L2	.TRUE.
L1.AND..NØT.L2.AND.L3	.TRUE.
L1.AND.L2.ØR.NØT.L3	.FALSE.
A.LT.B*C.AND.L2.ØR.I.NE.J	.TRUE.

II.1.7. Exerciţii

1. Care sînt etapele de rezolvare a unei probleme în FORTRAN ?
2. Care sînt particularităţile etapei de punere la punct a programelor ?
3. Care dintre exemplele de mai jos constituie constante în sensul definiţiilor date :

a) —121	b) 491.2	c) 514^2	d) 10*2
e) 41891	f) S418	g) ALFA	h) 15.
i) —563	j) —12 + 02	k) +12345678910	

4. Scrieţi următoarele cantităţi numerice sub formă de constante reale :

1 259,4 ; —2,959 ; 40001 ; 10^{25} ; 0,0000001 ; 10^{-7}.

5. Reprezentați aceste valori (exercițiul 4) sub formă de constante în precizie dublă.

6. Care dintre următoarele constante reale sînt scrise greșit ? De ce ?
a) 256,0001 b) —2E + 12 c) +4.3E1 — 3 d) —4.1E2.1

7. Care dintre perechile de constante reale au aceeași valoare ?

a) 517.9	+517.9		d) 23,1	+231E—1
b) 256.1	0.2561E+3		e) 8.0	8.
c) 0.1E—6	0.000001			

8. Care dintre următoarele constante în precizie dublă sînt corecte :

a) 4214D—05	d) 0.D2
b) 64.2156 DP—5	e) 0.D—4
c) —5.34.1D51	f) .D+2

Explicați cauza incorectitudinii celorlalte.

9. Ce valoare au următoarele constante complexe ?

(4.21, 1.0) (5.1E + 2, 1.5) (5.71E—1, —6.7)
(9.2E+2, —9.2E+02) (1., 2.) (4E+41, —5E—51)

10. Scrieți sub formă de constante complexe valorile :
4+2i, 5.2—6i, 1+i,
5.3—2.7162i, 4+i, —i.

11. Care dintre constantele complexe de mai jos nu sînt corecte :

a) (4.21, 5.1)	b) (2.E+1, 2)	c) (1,1)
d) (0., 0.)	e) (0.1E—86, 05E+90)	f) (0.1, 0)

12. Numele unor variabile FORTRAN de mai jos sînt greșit scrise. Care sînt ele și de ce ?

L1*—42	VALØARE 111	total
TØLERANȚA	K—22	VALØARE
25X	V—25	BUNA
ALFA	V_{MINIM}	A1B25

13. În programul în care apar numele variabilelor de mai jos nu se află nici o declarație cu privire la tip. Indicați care este tipul fiecăreia.

REZULTAT, REST, E, MAI, VAX, IJ,
SØRT, SACI, MARCA, X, Y, X1, XY,
EPSILØN, CICØ, PEPSI, BIG, TETA

14. Transcrieți algoritmii din capitolul I.2 folosind nume de variabile FORTRAN conforme cu tipul valorilor pe care le vor conține acestea.

15. Fără să folosiți declarații de tip, schimbați următoarele nume în așa fel încît să fie de tip întreg, păstrînd pe cît posibil sensul cuvintelor :

SUMA, CÎT, VAL, REST, GRAD, TEMP, X, Y.

16. Indicați de ce sînt greșite următoarele instrucțiuni FORTRAN :

a) ÎNTREG M1, ALFA, PIT,
b) DØUBLE PRECISIØN, ON, B,
c) REAL K2, A', IC.

17. Identificați erorile comise la scrierea următoarei secvențe de instrucțiuni :

```
REAL IØN, MAX
IMPLICIT INTEGER (A-M, Z)
IMPLICIT INTEGER (X—T)
```

18. De ce e greșită următoarea instrucțiune ?

```
IMPLICIT INTEGER(A—T), REAL(M)
```

19. De ce nu acceptă compilatorul aceste declarații?

```
|| REAL, M, N, V1
|| INTEGER, Z, V1, ALFA
|| CØMPLEX N, C1
```

20. Încercați să explicați de ce același compilator nu poate să lucreze pe două calculatoare diferite, deși programele FORTRAN au aceeași formă.

21. Într-un program s-a folosit următoarea declarație:

```
|| IMPLICIT INTEGER (A—Z)
```

Cum putem lucra totuși cu tablourile următoare:

A real cu 10 linii și 15 coloane
D1 și CM complexe cu 13 linii și 5 coloane
MINV logic cu 14 elemente?

22. Se consideră tabloul real Y cu 5 linii și 3 coloane.
a) Scrieți elementele acestuia parcurgînd coloanele una după alta.
b) Indicați ordinea elementelor cînd parcurgem prima linie apoi a doua etc. Coincide șirul găsit cu cel de la punctul a)?

23. Este posibil ca toate elementele de pe linia a 5-a din tabloul real cu 15 linii și 10 coloane să aibă valori complexe? De ce?

24. Identificați erorile din următoarele grupuri de instrucțiuni:

a) DIMENSIØN A(5, 1), B(9)
INTEGER MAX, A(6)
b) REAL A(4, 2, 3, 5, 6, 7, 8, 10)
c) DIMENSIØN ALFA (M, N)
d) DIMENSIØN, A, B (10, 7)

25. Scrieți în FORTRAN următoarele expresii (literele grecești reprezintă valori reale):

a) $(x+1)^{n+2}+x^{2n+1}$, b) $x^2-2x\cos\alpha+1$,

c) $\frac{a}{2-a}\cdot\frac{1+a}{2-b}$, d) abc,

e) $\frac{\frac{1}{b}-\frac{1}{a}}{\frac{m}{c}+ab}$, f) $\frac{-b+\sqrt{b^2-4ac}}{2a}$

g) $\frac{a}{2-a}\ (1-a)-\frac{1}{1-a}$, h) $\frac{a}{1-\frac{1}{b}}+\frac{1-\frac{1}{c}}{\frac{-1}{1-a}}+\frac{\frac{1}{x+y}}{2}$,

i) $\frac{-\cos^4\alpha\sin^3\alpha}{p+1}$, j) $\sqrt{p(p-a)(p-b)(p-c)}$.

26. Într-un program avem următoarele declarații:

```
|| IMPLICIT DØUBLE PRECISIØN (R—T), CØMPLEX (U—Z), LØGICAL (H, L)
|| DIMENSIØN F(40), L(7), TAU(15)
```

Arătați cum sînt evaluate expresiile de mai jos:

a) B1 + M1/M2**K*C—A*F(3)
b) A*B/C/D**M1(15)/F(16)*A2
c) 4.25*R2 + M1/M2*CAR**T1
d) MAX1(4.2, 5.E + 1,49.99)*ALFA/R1**M
e) U + V*W/(2.5,4.)**2
f) A.LT.R.AND..NØT.A.EQ.B**2 + 4*A*C
g) C*(1 — C)**2 + (— 2./3)/C + 7

27. Care sînt greşelile comise în scrierea următoarelor expresii (declaraţiile de tip sînt aceleaşi ca la exerciţiul precedent).

```
4M + N
A + C/*D + 7
```

28. Ce valori vor avea expresiile:

a) 4.2/2.1*1.5−3
b) 4.1*2*(2 /5)
c) 4.1*2*(2/5)
d) (3*(A**(1/3) + B**(1/2))/(A−B)**(6/10), unde A şi B au valorile 3.71, şi, respectiv, −2.73E + 5

29. Scrieţi în FORTRAN următoarele formule:

a) $t = \frac{1}{xy} + \frac{1}{yz} + \frac{1}{zx}$

b) $v = \frac{b+c}{a} + \frac{c+a}{b}$

c) $x = \frac{ab}{a+b} - \frac{a}{b+c}$

d) $y = \sqrt{a + \sqrt{ab}}$

e) $z = \left(\frac{\sqrt{ab - 5\,421}}{\sqrt{a} + \sqrt{e}}\right)^{2x}$

f) $z = \frac{a^{-\frac{1}{2}}}{69}\sqrt{\frac{\sin a}{2}}$

g) $x = \frac{y^2+2}{b}\sin y$

h) $y = x^a z^b$

i) $z = 5^{xy} + abc^a$

30. Fie declaraţia

```
|| IMPLICIT CØMPLEX (U−Z), DØUBLE PRECISIØN (P−T), LØGICAL (H, L)
```

Ce valori vor primi variabilele întrebuinţate în instrucţiunile de atribuire de mai jos?

```
|| A = 12*3 − 6
|| B = 16/2.
|| C = 2.*(12/3)
|| P = 5.*(1/5)
|| W = (2.2,5.15) + (4.1, 2.5)
|| U = (1.0,−1.)*(2., 1).
|| L1 = .TRUE. .AND. .FALSE.
```

```
|| IA = 5/2.
|| A11 = 5/2
|| A2 = 1/3
|| T = 20/6 + 15/2
|| UI = 5/2
|| I1 = 2*(5/2)
|| A1 = 1 000*(999/1 000)
|| M = 5.21D0 + 4.25
```

II.2. PRINCIPALELE INSTRUCŢIUNI ALE LIMBAJULUI FORTRAN

II.2.1. Introducere

Instrucţiunile «pseudolimbajului» LPS, studiate în capitolul precedent, cu ajutorul cărora s-au scris programe LPS, se pot grupa, în funcţie de prelucrarea îndeplinită, astfel[1]: instrucţiuni de intrare/ieşire, condiţionale, de ciclare, de atribuire şi de oprire.

În limbajul FORTRAN se folosesc pentru descrierea prelucrărilor tipuri de instrucţiuni similare cu cele de mai sus, dar care au o semantică şi o sintaxă diferită, proprie limbajului. Astfel, în LPS terminarea textului unei in-

[1] Pentru a înţelege mai bine cele ce urmează, este bine ca sintaxa şi semantica acestor instrucţiuni să fie reamintită.

strucţiuni de ciclare era indicată prin ■, în timp ce în FORTRAN nu avem un asemenea semn, ceea ce a făcut necesară, printre alte lucruri, introducerea şi folosirea etichetelor.

După cum am mai spus, etichetele pot fi utilizate în zona corespunzătoare a instrucţiunilor FORTRAN, atunci cînd programatorul consideră necesară identificarea instrucţiunii respective. De regulă, fiecare instrucţiune FORTRAN poate avea sau nu etichetă, lucru pe care îl presupunem implicit în descrierile instrucţiunilor date mai jos. Vom preciza explicit doar cazurile de excepţie : cînd apariţia unei etichete în zona corespunzătoare este obligatorie sau dimpotrivă interzisă. În continuare vom studia principalele tipuri de instrucţiuni FORTRAN.

II.2.2. Instrucţiunea condiţională

În FORTRAN sînt folosite două instrucţiuni condiţionale, prelucrările exprimate de acestea fiind mai puţin puternice decît ale instrucţiunilor corespunzătoare tip din LPS.

II.2.2.1. Instrucţiunea IF logic

Forma generală :

|| IF(*<expresie logică>*) *<s>*

unde *<s>* este orice instrucţiune executabilă cu excepţia lui DØ sau a unei alte instrucţiuni IF logic.

Semnificaţie. În momentul execuţiei acestei instrucţiuni, se evaluează valoarea expresiei logice. Dacă valoarea acesteia este «FALSE» atunci instrucţiunea *<s>* nu se execută continuîndu-se execuţia cu instrucţiunea următoare, în caz contrar se execută instrucţiunea *<s>*.

II.2.2.2. Instrucţiunea IF aritmetic

Forma generală :

|| IF (<expresie aritmetică>) $<e_1>$, $<e_2>$, $<e_3>$

unde $<e_1>$, $<e_2>$, $<e_3>$ sînt etichete.

Semnificaţie. Pentru execuţia acestei instrucţiuni se evaluează expresia aritmetică. În funcţie de valoarea expresiei (negativă, zero sau pozitivă), se execută instrucţiunea cu eticheta $<e_1>$, $<e_2>$ sau, respectiv, $<e_3>$.

Observaţie. Un rol important îl au în descrierea prelucrărilor, etichetele care permit să se indice locul unde se va continua prelucrarea în funcţie de valoarea unor expresii, de anumite condiţii etc.

II.2.3. Instrucţiuni de salt necondiţionat

Dacă într-o instrucţiune IF aritmetic toate etichetele sînt egale, execuţia programului se va continua cu aceeaşi instrucţiune. Astfel

IF(A — 2) 33, 33, 33

va avea drept efect continuarea execuţiei cu instrucţiunea avînd eticheta 33, indiferent de valoarea expresiei A — 2.

Deşi o asemenea posibilitate de programare poate fi folosită, ea nu este recomandată deoarece ar conduce la o înţelegere greoaie a prelucrării exprimate de program. Pentru a da posibilitatea programatorului să transfere controlul în orice punct al programului, limbajul FORTRAN îi pune la dispoziţie instrucţiunea GØ TØ.

Forma generală :

```
|| GØ TØ <e>
```

unde <*e*> este o etichetă.

Semnificaţie. După execuţia acestei instrucţiuni controlul va fi trecut (deci se va executa în continuare) instrucţiunii cu eticheta *e*.

Observaţie. Pentru a fi executată instrucţiunea care urmează după un GØ TØ sau un IF trebuie să aibă întotdeauna etichetă.

În secvenţa de instrucţiuni :

```
   || GØ TØ 72
   || IF(K) 73, 74, 75
72 || I = I + 1
```

instrucţiunea IF nu poate fi executată niciodată, ea fiind sărită întotdeauna datorită instrucţiunii GØ TØ. Pe de altă parte, nici o instrucţiune a programului nu poate cere execuţia ei, deoarece nu poate fi identificată (nu are etichetă).

II.2.4. Instrucţiunile : CONTINUE, STOP, PAUSE şi END

Forma generală a instrucţiunii CØNTINUE

```
|| CØNTINUE
```

Semnificaţie. Instrucţiunea nu afectează secvenţa normală de executare a instrucţiunilor. Cînd este întîlnită se trece la executarea instrucţiunii care o urmează în program.

Forma generală a instrucţiunii STØP

```
|| STØP
|| STØP <n>
```

unde <*n*> e o constantă întreagă, fără semn, cu cel mult 5 cifre.

Semnificaţie. Instrucţiunea pune în evidenţă sfîrşitul execuţiei programului. Utilizarea formei a II-a va produce afişarea pe consola calculatorului a numărului <*n*>. Cînd într-un program se folosesc mai multe instrucţiuni STØP, ştim astfel la care s-a ajuns, cînd a luat sfîrşit execuţia programului.

Forma generală a instrucţiunii PAUSE

```
|| PAUSE
|| PAUSE <n>
|| PAUSE <mesaj>
```

unde <*n*> e o constantă întreagă, fără semn, cu cel mult 5 cifre.

Semnificație. Se folosește pentru a opri temporar execuția unui program. Reluarea programului se face la comanda operatorului. Această instrucțiune se folosește fie pentru a cere o acțiune manuală a operatorului (schimbarea unei benzi, a unui disc etc.) fie pentru scrierea unor informații despre programul ce se execută (delimitarea unei erori etc.). Dacă se folosește forma a doua sau a treia, pe lîngă cuvîntul PAUSE care apare scris pe consolă, se adaugă și numărul <*n*>, respectiv mesajul indicat, astfel încît se poate identifica care a fost instrucțiunea PAUSE ce a întrerupt execuția programului.

Forma generală a instrucțiunii END

|| END

Semnificație. Este o instrucțiune neexecutabilă. Este folosită pentru a indica compilatorului care e ultima instrucțiune din programul sursă. Ea trebuie să apară în fiecare program. Această instrucțiune nu poate avea etichetă.

II.2.5. Instrucțiuni de intrare/ieșire

În LPS valorile variabilelor în forma externă erau reprezentate pe banda de intrare și banda de ieșire. Folosind instrucțiunea **citește** *x* valorile erau citite de pe banda de intrare și transferate în memoria principală. În mod similar valoarea unei variabile era transferată din memorie pe banda de ieșire prin instrucțiunea **scrie** *x*.

Pentru un calculator fictiv acest lucru era suficient. În cazul unui calculator real, datele pot fi reprezentate pe diverse medii externe ca, de exemplu : cartelă perforată, hîrtie de imprimantă, bandă magnetică sau disc magnetic. Aceasta complică foarte mult problema transferurilor de date spre și dinspre memoria principală.

Descrierea unui schimb de date între o unitate externă și memorie, deci a unei operații de intrare/ieșire, presupune precizarea :

— operației ce trebuie efectuată (citire sau scriere) ;

— variabilelor ale căror valori sînt implicate în transferul respectiv (prin aceasta este precizată și forma internă de reprezentare) ;

— locului unde se găsesc datele reprezentate în forma externă (operația de intrare/ieșire se va îndeplini între această unitate și memorie) ;

— formei de reprezentare externă a datelor (structura înregistrării care va fi citită sau scrisă).

Pentru ca operația de intrare/ieșire să fie cît mai independentă de dispozitivul fizic particular care va fi implicat în transferul de date, limbajul FORTRAN presupune că datele sînt organizate pe mediul extern sub formă de înregistrări. Mulțimea înregistrărilor de pe un mediu extern alcătuiește un set de date pe care adesea îl vom numi fișier. Fiecare set de date va primi un număr care-l va identifica. El va fi indicat în fiecare operație de intrare/ieșire. În momentul execuției se va preciza ce dispozitiv fizic corespunde acestui număr. *Exemplu :* dacă unui set de date pe cartele (înregistrarea este cartela de 80 de coloane) îi asociem numărul 105 atunci în toate operațiile de citire ce le vom îndeplini pentru acest set de date vom indica acest număr.

Fizic, setul de date se va prezenta sub forma unui teanc de cartele pe care sînt perforate unele date în formă externă. Abia în timpul execuţiei se va arăta pe ce cititor de cartele este plasat acest teanc.

Din punctul de vedere al limbajului FORTRAN, fiecare înregistrare se prezintă sub forma unui şir de caractere. Forma în care aceste caractere sînt grupate în cîmpuri (zone), care constituie reprezentările valorilor unor variabile, este precizată într-o descriere a structurii acestei înregistrări, dată prin instrucţiunea FORMAT. Această instrucţiune este ataşată unei instrucţiuni de ieşire/intrare.

Variabilele care participă la un transfer de date sînt precizate printr-o *listă de intrare/ieşire*. Deşi au aceeaşi formă, interpretarea lor este diferită în funcţie de operaţie : intrare sau ieşire.

Z(1, 1, 1)
Z(2, 1, 1)
Z(1, 2, 1)
Z(2, 2, 1)
Z(1, 3, 1)
Z(2, 3, 1)
Z(1, 1, 3)
Z(2, 1, 3)
Z(1, 2, 3)
Z(2, 2, 3)
Z(1, 3, 3)
Z(2, 3, 3)

Fig. II.9

O listă de intrare precizează variabilele simple sau indexate cărora le sînt atribuite valori printr-o operaţie de intrare.

O listă de ieşire precizează variabilele ale căror valori sînt luate din memorie şi scrise pe un mediu extern.

În listele de intrare/ieşire pot apărea nume de tablouri. În acest caz se presupune că toate elementele tabloului sînt implicate în operaţia respectivă, în ordinea aşezării lor în memorie. În FORTRAN elementele unui tablou sînt aşezate în locaţii succesive de memorie, mărindu-se la început valoarea primului indice pînă ajunge la dimensiunea maximă, apoi a celui de-al doilea etc. De exemplu, elementele tabloului Z (2, 3, 2) sînt plasate în memorie în ordinea din figura II.9. Dacă în lista de intrare/ieşire este indicat un element al unui tablou (variabilă indexată) atunci numai acel element participă la transfer.

II.2.5.1. Instrucţiunea de citire

Forma generală a instrucţiunii READ

READ (<*u*>, <*f*>, END = <*e*>) <*listă-intrare*>

unde <*u*> este numărul setului de date (pentru calculatorul FELIX-C numărul 105 desemnează unitatea standard de intrare — cititorul de cartele) ;

<*f*> — eticheta instrucţiunii FORMAT asociată instrucţiunii READ ;

<*e*> — eticheta unei instrucţiuni de la care se va continua prelucrarea în cazul în care s-a cerut citirea valorilor unor variabile, dar în setul de date nu mai sînt înregistrări.

Semnificaţie : Prin execuţia acestei instrucţiuni, sînt atribuite valori variabilelor din lista <*listă-intrare*>, valori care sînt citite din una sau mai multe înregistrări ale setului de date cu numărul <*u*>. Caracteristicile valorilor şi aranjarea lor în înregistrările de pe mediul extern sînt descrise în instrucţiunile FORMAT cu eticheta <*f*>. Folosirea opţiunii END = <*e*> nu este obligatorie ; dacă ea nu este folosită şi nu se pot atribui valori tuturor variabilelor din listă-intrare, deoarece nu mai sînt înregistrări pe setul de date <*u*>, este semnalată o eroare şi prelucrarea se opreşte ; în situaţia cînd este folosită, se continuă prelucrarea cu instrucţiunea cu eticheta <*e*>.

II.2.5.2. Instrucțiunea de scriere

Forma generală a instrucțiunii WRITE

WRITE (*<u>*, *<f>*) *<listă-ieșire>*

unde : *<u>* reprezintă numărul setului de date de ieșire (pentru calculatorul FELIX-C numărul unității standard de ieșire — imprimanta — este 108) ;
<f> — eticheta instrucțiunii FORMAT asociate ;
<listă-ieșire> — lista de ieșire.

Semnificație : Variabilele din lista de ieșire <listă-ieșire> vor fi scrise pe setul de date cu numărul *<u>*, sub formă de înregistrări al căror format este specificat în instrucțiunea FORMAT cu eticheta *<f>*.

Exemplu. În acest moment avem cunoștințele necesare pentru a înțelege programul din figura II.2. Acest program, după cum arată comentariul din linia 10 (vezi zona de identificare), găsește soluția unei ecuații de gradul I. Algoritmul după care se rezolvă ecuația cu coeficienții A și B este cel din figura I.20, descris în LPS. Vom descrie pe scurt liniile programului FORTRAN. După declararea variabilelor reale A, B și X (linia 20) urmează citirea coeficienților A și B ai ecuației (linia 30) dintr-o cartelă folosind instrucțiunea FORMAT cu eticheta 11. În continuare (linia 40) prin instrucțiunea IF logic se testează dacă valoarea lui A este zero ; dacă acest lucru este adevărat se sare necondiționat la instrucțiunea cu eticheta 1. Dacă A nu are valoare nulă, se continuă cu calculul soluției X (linia 50), scrierea ei pe imprimantă cu FORMAT-ul 12 (linia 60) și saltul necondiționat prin instrucțiunea GØ TØ la instrucțiunea cu eticheta 3 care indică sfîrșitul prelucrării (linia 120) deoarece ecuația a fost rezolvată. Dacă B = 0 se sare la instrucțiunea cu eticheta 2, unde se scrie pe imprimantă că avem o infinitate de soluții, după care se oprește prelucrarea. Ultima secvență, care se execută cînd B e diferit de zero, afișează pe imprimantă mesajul «nici o soluție». O ultimă observație este că între sfîrșitul prelucrării (linia 120) și sfîrșitul textului programului (linia 170) se află mai multe instrucțiuni neexecutabile.

Exercițiu. Căutați să identificați, în descrierea de mai sus, liniile sursă care descriu prelucrările arătate, dar ale căror numere nu au fost precizate. Identificați elementele componente ale fiecărei instrucțiuni cunoscute.

II.2.6. **Instrucțiunea FORMAT**

Instrucțiunea FØRMAT este o instrucțiune neexecutabilă, care dă informații despre structura înregistrărilor dintr-un set de date. În felul acesta compilatorul va deduce unde și cum sînt reprezentate valorile variabilelor din lista de intrare/ieșire în înregistrare. O înregistrare se compune din mai multe cîmpuri, fiecare cîmp putînd conține forma externă a unei valori numerice sau alte date (texte, spații etc.). Instrucțiunea FØRMAT va trebui să ofere informații despre fiecare din cîmpurile înregistrării. Ea face acest lucru prin intermediul descriptorilor. De asemenea, prin intermediul unei instrucțiuni se pot descrie mai multe înregistrări.

Forma generală a instrucțiunii FØRMAT este :

<etichetă> || FØRMAT (*<listă-descriptori>*)

Semnificație : Descriptorii care apar în <listă-descriptori> (numită în continuare și lista FØRMAT) vor arăta care este structura unei înregistrări. Aceste instrucțiuni pot apărea oriunde în program (nu înainte de instrucțiunea IMPLICIT sau după instrucțiunea END) și trebuie să aibă etichetă. Între lista de intrare/ieșire și lista de descriptori se stabilește o corespondență astfel : fiecărei variabile din lista de intrare/ieșire i se asociază un descriptor de reprezentare a valorii acesteia.

În continuare, vom analiza următoarele categorii de descriptori : de reprezentare a valorilor variabilelor, de așezare a cîmpurilor în înregistrare și a înregistrărilor în cadrul setului de date.

II.2.6.1. Descriptori de reprezentare a valorilor variabilelor

Fiecărei variabile sau fiecărui element de tablou din lista de intrare/ieșire trebuie să le corespundă în lista FØRMAT cîte un cod care va descrie cîmpul din înregistrare de unde se citesc sau unde se scriu valorile asociate.

Vom descrie în continuare descriptorii utilizați pentru manipularea valorilor întregi și reale. Pentru a insera explicațiile la descrierea fiecărui descriptor, presupunem că variabila căreia acesta i-a fost asociat este *var*, iar cîmpul în care valoarea este reprezentată ocupă *w* poziții.

● *Descriptorul* I *pentru transmiterea valorilor variabilelor de tip întreg*
Forma generală : I*w*
Semnificație — litera I arată că variabila *var* căreia i s-a asociat acest descriptor trebuie să fie de tip întreg.

La intrare valoarea din cîmpul de *w* poziții al înregistrării de intrare este citită, se efectuează conversiunea la forma internă și rezultatul este atribuit variabilei *var*. Șirul de caractere din cîmpul de intrare trebuie să aibă forma unei constante întregi cu sau fără semn. Dacă semnul nu este pus, valoarea se consideră pozitivă. Dacă în acest cîmp apar spații ele sînt considerate zerouri.

La ieșire : valoarea variabilei întregi corespunzătoare din lista de ieșire va fi afișată sub forma unui număr întreg, zecimal, aliniat la dreapta în cîmpul de *w* poziții.

Afișarea se face cu semnul — dacă valoarea e negativă și fără semn dacă e pozitivă. Dacă cele *w* poziții nu sînt suficiente pentru a afișa valoarea variabilei *var* în formă externă, în cîmpul destinat aflării vor fi scrise asteriscuri.

Exemplul 1. Dacă variabila K are valoarea +1705 și folosim pentru afișare formatul I7 va avea reprezentarea din figura II.10.

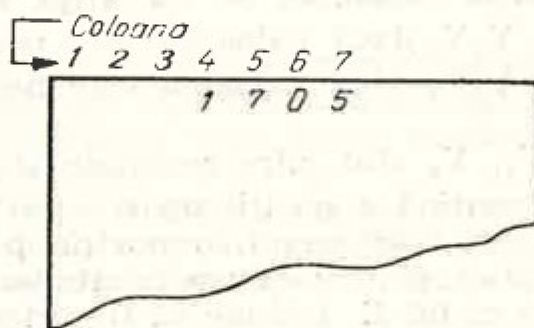


Fig. II.10

Exemplul 2. În figura II.11 sînt reprezentate diferite imagini ale unor cartele de pe care se pot citi, cu codul indicat în dreptul acestora, valori egale pentru variabilă : —70209.

Cod folosit	Imaginea primelor coloane ale cartelei
I6	`-70209`
I7	` -70209`
I7	`- 70209`
I7	`- 7 209`
I8	`-  70209`
I8	` - 7 2 9`
I9	`  - 702 9`

Fig. II.11

● *Descriptorii pentru transmiterea valorilor variabilelor reale și precizie dublă*

Cîmpul unde este reprezentată valoarea unei astfel de variabile se poate descrie cu oricare din codurile E, F sau D.

Forma generală a acestor coduri este :

$$Fw.d \quad Ew.d \quad Dw.d$$

unde w indică cîte caractere are întregul cîmp în care este reprezentată valoarea, iar d numărul cifrelor zecimale din reprezentare. Este evident că trebuie să avem $w \geqslant d$.

Semnificația de intrare : Reprezentarea valorii variabilei în cele w poziții se poate face folosind sau nu punctul zecimal. Folosind punctul zecimal se reprezintă valoarea sub forma unei constante reale. Toate blancurile ce apar în reprezentare sînt considerate drept zerouri.

— *Dacă nu folosim punctul zecimal în reprezentarea valorilor ce vor fi citite, o importanță hotărîtoare o va avea indicația dată de d.* Astfel, cînd în cîmpul de w poziții se află un șir de cifre zecimale cu sau fără semn și fără exponent zecimal, valoarea transmisă va fi găsită considerînd că ultimele d poziții din cîmp constituie partea zecimală.

— *Dacă apare un exponent zecimal, iar șirul de cifre care îl precede nu conține punctul zecimal, atunci primele d poziții din fața exponentului sînt considerate ca alcătuind partea zecimală.*

Semnificația la ieșire : Afișarea valorii oricărei variabile reale sau în precizie dublă se poate face cu oricare din cele trei coduri, E, F, D. După ce valoarea a fost convertită în zecimal, reprezentarea va avea o formă ce depinde numai de codul folosit.

Descriptorul E. Valoarea variabilei se va afișa sub una din formele :

$X_1X_2X_3 \ldots X_dE \pm Y_1Y_2$ dacă valoarea este pozitivă

$-X_1X_2X_3 \ldots X_dE \pm Y_1Y_2$ dacă valoarea este negativă

unde :

$X_1, X_2, \ldots, X_d, Y_1, Y_2$ sînt cifre zecimale și $X_1 \neq 0$.

Așadar, w trebuie să conțină 4 poziții pentru partea de exponent zecimal, d poziții pentru partea fracționară, o poziție pentru punctul zecimal și o poziție pentru semn. Deci, pentru ca reprezentarea externă a valorii unei variabile să fie posibilă în codul E, trebuie să fie adevărată relația :

$$w \geqslant d + 6.$$

Exemplu. Fie secvenţa :

```
   || ALFA = - 409.79
   || WRITE (4,25) ALFA
25 || FØRMAT (F11.5)
```

Prin aceasta este scrisă pe unitatea externă identificată cu numărul 4 o înregistrare alcătuită din 11 caractere, cu următoarea formă :

$$-.40979E+03$$

Numărul se scrie aliniat la dreapta în cele w poziţii. Poziţiile neocupate $(w-d-6)$ se completează cu blancuri.

Descriptorul F. Reprezentarea valorii variabilei se face în w poziţii ultimele d conţinînd partea fracţionară. Ea este separată de partea întreagă prin punctul zecimal.

Descriptorul D. Reprezentarea se face normalizat, sub una din formele următoare :

$$.X_1X_2...X_dD \pm Y_1Y_2$$

$$-.X_1X_2...X_dD \pm Y_1Y_2$$

unde $X_1, \ldots, X_d$ sînt cifre zecimale, iar Y_1Y_2 constituie partea de exponent, $X_1 \neq 0$.

● *Descriptori pentru transmiterea valorilor variabilelor complexe*

Transmiterea unei valori complexe se face folosind doi descriptori de tipul F, E sau D, unul pentru partea reală şi altul pentru partea imaginară.

Exemplu. Dacă într-un program se foloseşte variabila complexă C pentru a citi valoarea ei de pe o cartelă se poate folosi instrucţiunea :

```
44 || FØRMAT (F7.3, E11.5)
```

● *Descriptorul* L *pentru transmiterea valorilor variabilelor logice*

Forma generală : Lw

Semnificaţie : În cele w caractere la intrare, valorile logice adevăr şi fals se reprezintă prin literele T şi respectiv F, scrise în orice poziţie a cîmpului. La ieşire, reprezentarea valorilor logice se face prin T sau F scrise în ultima poziţie a cîmpului.

II.2.6.2. Aşezarea cîmpurilor în înregistrare şi a înregistrărilor în cadrul setului de date

Modul în care apar valorile variabilelor în înregistrările de intrare/ieşire nu este indiferent. De multe ori este necesară aranjarea valorilor variabilelor într-un anumit mod, intercalarea unor texte suplimentare care să explice despre ce valori este vorba etc.

Programatorul poate folosi pentru a descrie poziţia cîmpului în înregistrare şi a unei succesiuni de înregistrări, codurile H'' X, caracterul de control al imprimării şi separatorului /. Aceste coduri nu se asociază unor variabile din lista de intrare/ieşire.

Descriptorul X

Forma generală : nX

Semnificație la intrare: ignorarea a *n* caractere consecutive; *la ieșire:* inserarea a *n* spații între două zone ale înregistrării[1].

● *Descriptorul de afișare a textelor*

Forma generală: '<*text*>'

Semnificație. Acest cod permite ca la ieșire să fie afișat textul indicat între apostrofuri[1].

Exemplu. Instrucțiunile:

```
   X = 2.5
   X = 4*X + 3.256
   WRITE (108, 22)X, Y
22 FØRMAT ('⊔⊔X = ',F9.2,2X,'F(X) = ',E12.3)
```

vor produce afișarea la imprimantă a unui text de forma:

```
⊔X =⊔⊔⊔⊔⊔2.50⊔⊔F(X) =⊔⊔⊔⊔.132E+0.2
```

Observație. Primul caracter a fost utilizat pentru controlul imprimării.

Forma generală: $n\mathrm{H}c_1c_2\ldots c_n$

Semnificație: la ieșire este imprimat pe *n* poziții textul $c_1c_2\ldots c_n$ care urmează după H[1]. Are același efect ca și codul precedent.

Exemplu. Instrucțiunea FØRMAT din exemplul precedent ar putea avea forma următoare, efectul rămînind neschimbat.

```
    WRITE (108, 122)X, Y
122 FØRMAT (4H⊔⊔X =,F9.2,2X,5HF(X)=,E12.3)
```

avînd însă același efect.

● *Caracterul de control al imprimării*

Un rînd scris pe imprimantă poate fi alcătuit din cel mult 132 de caractere. Pentru ca așezarea rîndurilor pe pagina de imprimantă să poată fi hotărîtă de programator, s-a convenit ca primul caracter din înregistrarea de ieșire să nu fie imprimat, el dînd informațiile necesare în acest sens.

La descrierea unei înregistrări pentru ieșire, primul caracter este caracter de control al imprimării și are una din formele:

0 — avans cu două linii înaintea imprimării,

1 — avans la prima linie a paginii următoare,

\+ nici un avans.

Blanc sau orice caracter diferit de 0, 1 și + înseamnă avans cu o linie înaintea imprimării.

Caracterul de control poate fi specificat în lista **FØRMAT** astfel: 1 H*x* sau '*x*', unde *x* are una din cele patru forme menționate anterior.

● *Separatorul / (slash)*

Utilizarea separatorului / (slash) în <listă-descriptori> are următoarele efecte:

— *Dacă la începutul sau sfîrșitul listei FØRMAT apar n slash-uri, atunci la intrare vor fi sărite n înregistrări, iar la ieșire n înregistrări vor fi pline cu* blancuri.

---

[1] În cazul în care dispozitivul de ieșire este imprimanta, primul caracter al înregistrării nu se afișează.

Exemplu :

```
   || REAL A(5)
   || READ (105, 33) (A(I), I = 1,5)
33 || FØRMAT (///5F7.2)
```

La execuție, vor fi sărite trei cartele (înregistrări), valorile elementelor tabloului fiind citite de pe a 4-a cartelă.

— Cînd apare un slash în interiorul unei liste FØRMAT se trece la înregistrarea următoare. Prin utilizarea a n slash-uri consecutive în interiorul unei <listă-descriptori> vor fi sărite la intrare n — 1 înregistrări, sau se vor introduce n — 1 înregistrări cu spații, la ieșire.

Observație — Cînd slash-ul separă 2 coduri în lista FØRMAT, nu se mai folosesc virgulele.

11.2.7. Reguli de alcătuire a listei FORMAT

Lista FØRMAT este formată din descriptori care descriu structura înregistrării.

În alcătuirea acestei liste pot fi folosite următoarele posibilități :

a) Dacă un descriptor d se repetă de k ori consecutiv atunci poate fi folosită în listă forma echivalentă kd.

Exemplul 1. Instrucțiunea

```
14 || FØRMAT (I4, I4, I4, F5.2, I4)
```

este echivalentă cu următoarea

```
14 || FØRMAT (3I4, F5.2, I4)
```

unde 3 indică numărul de repetări al descriptorului I4.

b) Dacă un grup de descriptori $d_1, d_2, \ldots, d_i$ se repetă în aceeași ordine de k ori, atunci în descrierea *<listă-descriptori>* se poate folosi un element de forma $k(d_1, d_2, \ldots, d_i)$.

Exemplul 2. Instrucțiunea :

```
51 || FØRMAT (I3, I4, E11.5, I4, E11.5, D14.6)
```

este echivalentă cu următoarea :

```
51 || FØRMAT (I3, 2(I4, E11.5), D14.6)
```

11.2.8. Reguli după care se explorează lista FORMAT

Pentru fiecare variabilă care apare în lista de intrare/ieșire se stabilește care este descriptorul corespunzător din *<listă-descriptori>* (pe care o vom desemna și sub numele echivalent de listă FØRMAT), descriptor care precizează reprezentarea externă a valorii variabilei.

Fie n numărul variabilelor şi al elementelor de tablou din lista de intrare/ieşire. În lista instrucţiunilor FØRMAT asociată, presupunem că se află m descriptori pentru transmiterea valorilor variabilelor.

În funcţie de valorile lui *m* şi *n* desprindem următoarele acţiuni :

a) *n = m — Lista de intrare/ieşire se poate satisface doar printr-o singură parcurgere a listei FØRMAT. Fiecărei variabile sau element de tablou îi corespunde un singur descriptor din lista FØRMAT.*

(b) *n < m — Lista de intrare/ieşire poate fi satisfăcută fără să folosim toţi descriptorii de tratare a variabilelor din lista FØRMAT. În acest caz sînt folosiţi doar primii n descriptori.*

Exemplul 1.

```
   || READ (1, 33)I, J, K
33 || FØRMAT (I6, I7, I5, I4, I2)
```

Descriptorii I4, I2, nu sînt utilizaţi :

(c) *n > m — Toate variabilele din lista de intrare/ieşire nu pot fi satisfăcute printr-o singură explorare a listei FØRMAT. Pentru a se citi/scrie valorile tuturor variabilelor, se va relua explorarea formatului citindu-se/scriindu-se o nouă înregistrare ;*

— Explorarea se reia de la început, dacă în listă nu apar grupuri de descriptori închise între paranteze. În cazul cînd apar astfel de grupuri, explorarea se reia de la ultimul grup folosit. O regulă practică de găsire a acestuia este : se ia grupul a cărui paranteză este ultima închisă în dreapta (parantezele ce închid lista FØRMAT nu se iau în considerare).

Exemplul 2. Fie instrucţiunile :

```
   || WRITE (3,21)X, Y, Z, V, W, T
21 || FØRMAT (2F2.1, 3F7.2)
```

Explorarea se reia de la început. În prima înregistrare vor fi scrise valorile variabilelor X, Y, Z, V, W iar în a doua, doar valoarea lui T descriptorul fiind F2.1.

Dăm mai jos alte forme ale instrucţiunii, precum şi punctele din care se reia explorarea, pentru aceeaşi listă de variabile.

```
21 || FØRMAT (2F2.1, 3(F7.2))
                       ↑
              reluarea explorării

21 || FØRMAT (F2.1, (F2.1, 3F7.2))
                    ↑
              reluarea explorării

21 || FØRMAT (F2.1, (F2.1, F7.2), 2F7.2)
                    ↑
              reluarea explorării
```

II.2.9. Exemple de programe FORTRAN

Exemplul 1. Programul de rezolvare a ecuației de gradul I, din figura II.2, poate fi acum înțeles pe deplin. Instrucțiunile cu numerele 130 pînă la 160 descriu structura înregistrărilor de intrare și de ieșire. Căutați să lămuriți rolul fiecărei părți a acestor instrucțiuni.

```
      INTEGER R, T
      READ (105, 1) M, N
1     FØRMAT (2I6)
      IF(M.GE.N) GØTØ 100
        T = N
        N = M
        M = T
100   CØNTINUE
101     R = M - (M/N)*N
        M = N
        N = R
      IF(R.NE.0) GØTØ 101
      WRITE (108, 2) M
2     FØRMAT (1, 'C.M.M.D.C.=', I6)
      STØP
      END
```

Fig. II.12

Exemplul 2. Algoritmul lui Euclid (programul LPS, cap. I.3.2). Programul corespunzător în limbajul FORTRAN este cel din figura II.12 : în acest program valorile variabilelor M și N sînt citite de pe o cartelă, T și R sînt declarate de tip întreg. Analizați programul FORTRAN insistînd în mod deosebit asupra instrucțiunii cu eticheta 101.

Din exemplele de programare de mai sus se remarcă modul în care au fost codificate în FORTRAN instrucțiunile limbajului LPS. În figura II.13 este arătată schema după care recomandăm să se codifice instrucțiunea condițională, iar în figura II.14 schemele

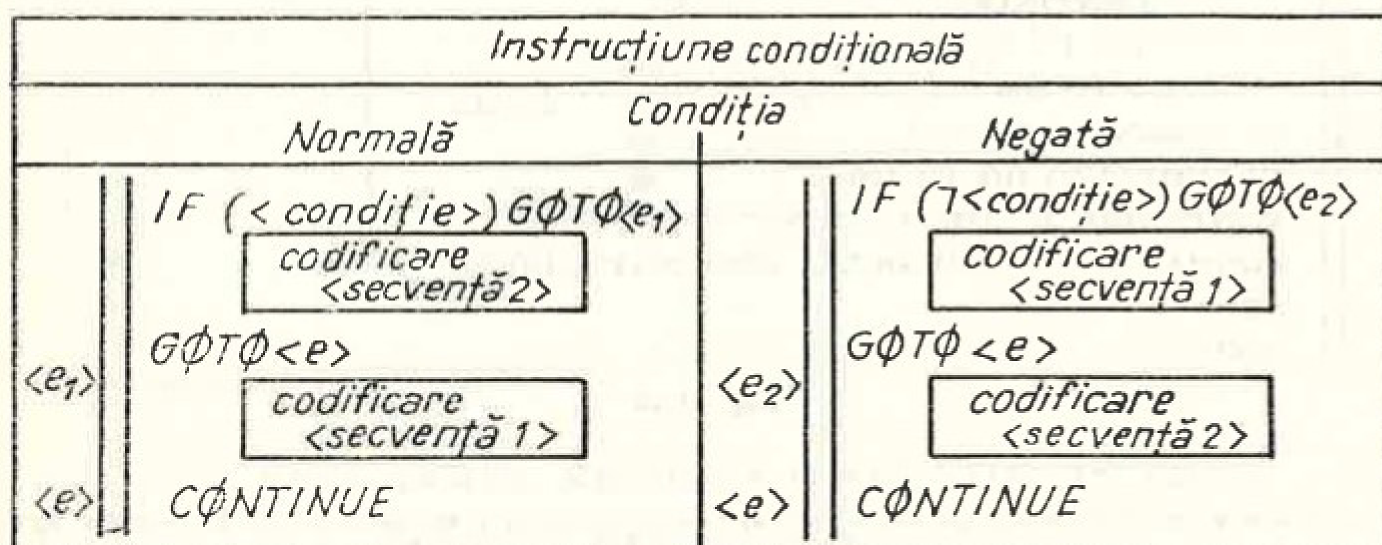


Fig. II.13

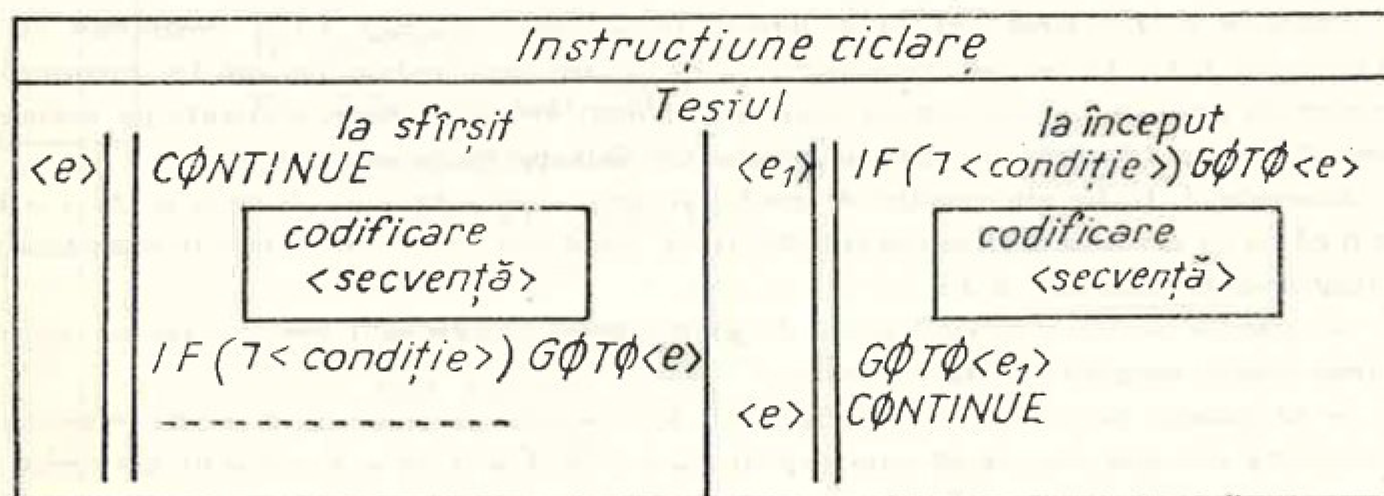


Fig. II.14

```
      INTEGER A(1000), T
      LØGICAL SEM
      READ (105, 1) N
 1    FØRMAT (I3)
      READ (105,2) (A(K), K = 1, N)
 2    FØRMAT (8I10)
100   CØNTINUE
         SEM = .TRUE.
         I = 1
200      IF(I.GT.N-1) GØ TØ 201
           IF(A(I).GE.A(I + 1))GØ TØ 301
             T = A(I)
             A(I) = A(I + 1)
             A(I + 1) = T
             SEM = .FALSE.
301        CØNTINUE
           I = I + 1
         GØ TØ 200
201      CØNTINUE
      IF (.NØT.SEM) GØ TØ 100
      WRITE (108, 3) (A(I), I = 1, N)
 3    FØRMAT (T 30, 'TABLØUL A ØRDØNAT' /(10I12))
      STØP
      END
```

Fig. II.15

corespunzătoare pentru instrucţiunile de ciclare cu test iniţial şi cu text final. În aceste figuri s-au respectat notaţiile din LPS iar etichetele FORTRAN sînt notate cu <e$_1$>, <e$_2$> şi <e>. Precizăm că ori de cîte ori instrucţiunea CØNTINUE nu este necesară, ea nu va fi folosită.

Exemplul 3. Ordonarea unui fir de numere (algoritmul LPS, cap. I.3.3). Programul este dat în figura II.15. Au fost folosite schemele de codificare recomandate mai sus. De asemenea, precizăm că programul poate ordona şiruri de cel mult 999 de numere, perforate pe cartele, numărul acestora, N, fiind perforat în primele trei coloane ale primei cartele.

Exemplul 4. Rădăcinile ecuaţiei de gradul al doilea $ax^2 + bx + c = 0$ unde a, b, $c \in R$ pot fi găsite cu ajutorul unui program FORTRAN. Condiţiile care trebuie luate în considerare la alcătuirea programului sînt :

— valorile coeficienţilor sînt citite de pe o cartelă ; coeficienţii vor fi scrişi cu punct zecimal fiecare ocupînd cîte zece coloane :

— în funcţie de valorile coeficienţilor a, b, c ecuaţia va avea două soluţii, o soluţie, o infinitate sau nici una, după cum ($a \neq 0$), ($a = 0$ şi $b \neq 0$, ($a = b = c = 0$) şi respectiv ($a = b = 0$ şi $c \neq 0$) ; pe imprimantă se cere să se afişeze ce ecuaţie s-a rezolvat şi rădăcinile acesteia, în cazul în care acestea există.

Descrierea algoritmului în LPS este dată în continuare.

```
citeşte a, b, c

| dacă a = 0
|       atunci | dacă b = 0 | dacă c = 0
|              |   atunci   |           atunci scrie 'ecuaţie nedeterminată'
|              |            |           scrie 'soluţie este orice x ∈ R'
|              |            |           altfel scrie 'ecuaţie imposibilă'
|              |            ■
|              | altfel atribuie x ← —c/b
|              ■
|              | scrie 'ecuaţie gradul I', a, b, c, x
|   altfel atribuie d ← b² — 4*a*c
|              | dacă d > 0
|              |   atunci atribuie x1 ← (—b + √d)/(2*a)
|              |                   & x2 ← (—b — √d)/(2*a)
|              |          scrie 'ecuaţie gradul II, rădăcini reale'
|              |          scrie a, b, c, x1, x2
|              |   altfel atribuie d ← —d
|              |          & x real ← —b/(2*a) & x imag ← √d/(2*a)
|              |          ■
|              |  scrie 'ecuaţie gradul II, rădăcini complexe'
|■             |  ■ scrie a, b, c, x real, x imag.
```

Programul FORTRAN este:

```
C        PROGRAMUL REZOLVĂ O ECUAŢIE DE TIPUL
C                A*X**2 + B*X + C = 0
C        UNDE A, B, C, SÎNT CITIŢI DE PE CARTELE
         READ (105, 100) A, B, C
         IF (A.NE.O) GØ TØ 200
         IF (B.NE.O) GØ TØ 21
            IF (C.NE.O) GØ TØ 11
               WRITE (108, 101)
              GØ TØ 99
11            WRITE (108, 102)
              GØ TØ 99
21            X = —C/B
              WRITE (108, 103) A, B, C, X
              GØ TØ 99
```

```
200          DELTA = B**2 - 4*A*C
             IF(DELTA.LT.0) GØTØ 211
                X1 = (-B + SQRT (DELTA))/(2*A)
                X2 = (-B - SQRT (DELTA))/(2*A)
             WRITE (108, 104) A, B, C, X1, X2
             GØTØ 99
211             XREAL = - B/(2*A)
                XIMAG = SQRT (-DELTA)/(2*A)
                WRITE (108, 105) A, B, C, XREAL, XIMAG
             GØTØ 99
99     STØP
100    FØRMAT (3 F10.3)
101    FØRMAT ('␣', 'ECUAȚIE NEDETERMINATĂ' /
     + '␣ØRICE X DIN R ESTE SOLUȚIE')
102    FØRMAT ('␣ECUAȚIE IMPOSIBILĂ')
103    FØRMAT ('␣ECUAȚIE DE GRADUL I', 4F10.3)
104    FØRMAT ('␣ECUAȚIE DE GRADUL DØI, RĂDĂCINI
     + REALE' / '␣', 3F10.3'/'␣', 2F10.3)
105    FØRMAT ('␣ECUAȚIE DE GRADUL DØI, RĂDĂCINI'
     +        '␣CØMPLEXE' /'␣', 3 F10.3/'␣', 2F10.3)
       END
```

II.2.10. Exerciții

1. Scrieți în limbajul FORTRAN următoarele fraze:
   a) Dacă $a \geqslant b$ valoarea lui x e egală cu 15.6, altfel e egală cu −21.5.
   b) Dacă $\alpha + \beta$ e pozitiv atunci $y \leftarrow 15.7$ altfel dacă $x < 7$ atunci $y \leftarrow -21$ altfel $y \leftarrow 121.1$
   c) Dacă $\alpha y - 5$ e egală cu zero execută în continuare instrucțiunea cu eticheta 129, altfel execută instrucțiunea următoare, cu numărul 767.

2. Scrieți în FORTRAN:
   a) Dacă $a < 0$ și $b > 0$ sau dacă $\alpha = 0$ atunci $x \leftarrow -2$.
   În caz contrar valoarea lui x nu se schimbă.
   b) Dacă $10^{-6} < x \leqslant 1$, se execută instrucțiunea cu eticheta 777, altfel se oprește procesul de calcul (STOP).

3. Se dă valoarea lui K. Să se descrie în limbajul FORTRAN următoarea prelucrare: Dacă valoarea lui K este pară se execută instrucțiunea cu numărul 615, dacă e impară se continuă prelucrarea.

4. Variabila RAD conține valoarea unui unghi exprimată în radiani. Alcătuiți schema logică și programul după a căror execuție variabilele întregi GR, MIN și SEC, conțin numărul de grade, minute și, respectiv, secunde corespunzătoare unghiului RAD.

5. Care e rezultatul lui S după executarea următoarelor programe:

a)

```
       I = 1
6      P = 7*1
       IF(I-4)2, 3, 2
2      I = I + 1
       GØ TØ 6
3      S = P
       WRITE (108, 1)S
1      FØRMAT ('␣', F7.2)
       STØP
       END
```

b)

```
       S = 7.5
7      IF(S - 10)5, 6, 6
5      S = S*1.2
       GØ TØ 7
6      S = S + 1
       WRITE (108, 1)S
1      FØRMAT ('␣',F7.2)
       STØP
       END
```

6. Identificați erorile comise în scrierea următoarelor instrucțiuni :

```
GØ TØ K
IF(K — 5) 6, 6, 67.
V + 5 = 2 + IVAR
IF(K*5) — 6) 5.2, 3
I + 7 = ALFA
IF(VAX)6, 6, 3, 2
1 VALOARE = ABCDEIFGH
```

7. Scrieți programul în limbajul FORTRAN care să citească de pe o cartelă valoarea variabilei N și să afișeze apoi valoarea variabilei S unde

$$S = 1 + 2 + \ldots + n.$$

Indicație. Calculați pe S fără să folosiți formula $S = \frac{n(n+1)}{2}$.

8. Scrieți un program în limbajul FORTRAN care să găsească numerele prime între 1 și N unde N este citit de pe o cartelă.

9. Indicați toate variabilele și elementele de tablou care fac parte din listele instrucțiunilor READ/WRITE, de mai jos :

```
REAL A(5), B(5.7), C(49)
INTEGER M (17)
DØUBLE PRECISIØN F (10, 20), D(4, 2, 5)
CØMPLEX C (21)
READ (105, 78) (A(I), I = 1, 5, 2)
READ (105, 79) A(2), ((B(L, M), M = 1, 7  3), L = 1, 4)
READ (105, 80) A(6), (B(2, 4), ((C(IV), IV = 4, 21, 4)
WRITE (108, 21) (((D(I, J, K), J = 1, 2), I = 1, 4), K = 1, 5, 2)
WRITE (108, 22) A, ((B(I, J), J = 1, 3), I = 1, 5)
WRITE (108, 23) ((F(L, M), L = 4, 8), M = 5, 7), (C(1), C(7))
```

II.3. FACILITĂȚI DE PROGRAMARE ÎN LIMBAJUL FORTRAN

Cu instrucțiunile studiate pînă în prezent se poate descrie în FORTRAN aproape orice prelucrare. Totuși limbajul oferă multe posibilități de programare, importante, dintre care o parte o vom studia în continuare : instrucțiunea DØ și subprogramele.

II.3.1. Instrucțiuni DO

Această instrucțiune este replica în FORTRAN a instrucțiunii de ciclare cu contor din limbajul LPS. Datorită asemănării lor vom da mai puține explicații asupra prelucrărilor exprimate de aceasta.

Formă generală :

DØ *n contor* = *vinit, vfin, vpas*

unde :

— *n* este eticheta unei instrucțiuni executabile ce urmează în program după instrucțiunea DØ ;

— *contor* se numește variabila ciclului și poate fi un nume de variabilă întreagă (nu poate fi element de tablou) ;

— *vinit, vfin, vpas* se numesc parametrii ciclului ; ei sînt fie variabile de tip întreg cu valori pozitive (nu elemente de tablou), fie constante pozitive fără semn.

În cazul cînd valoarea lui *vpas* este 1, instrucțiunea se scrie astfel :

DØ *n contor* = *vinit, vfin*

Este necesar să avem $0 < vinit \leqslant vfin$ și $vpas > 0$.

Semnificație : Executarea acestei instrucțiuni începe prin a atribui variabilei *contor* valoarea lui *vinit* după care :

— se execută toate instrucțiunile care urmează după DØ pînă la instrucțiunea cu eticheta *m* inclusiv (acest grup de instrucțiuni se numește uneori corpul ciclului DØ) ;

— valoarea variabilei *contor* este mărită cu aceea a lui *vpas* (cînd *vpas* lipsește, cu 1) ;

— dacă valoarea lui *contor* depășește pe aceea a lui *vfin* atunci se continuă execuția programului cu instrucțiunea care urmează imediat după aceea cu eticheta *n*, altfel se vor aplica din nou regulile de mai sus.

Exemplu. Înmulțirea a două matrice. În FORTRAN programul va fi următorul :

```
      INTEGER P
      REAL A (20, 15), B(15, 10), C(20, 10)
C**** CITIREA DIMENSIUNILOR REALE ȘI A MATRICELOR**
      READ (105,1) N, M, P
      READ (105,2) ((A(I, J), I = 1, N), J = 1, M)
      READ (105,2) ((B(I, J), I = 1, M), J = 1, P)
C**** ÎNMULȚIREA MATRICELOR
        DØ 11 I = 1, N
          DØ 12 K = 1, P
            S = 0
            DØ 13 J = 1, M
                S = S + A(I, J)*B(J, K)
13          CØNTINUE
            C(I, K) = S
12        CØNTINUE
11      CØNTINUE
C***  AFIȘARE REZULTATE
      WRITE (108,3) ((C(I, J), J = 1, P), I = 1, N)
      STØP
1     FØRMAT (3I5)
2     FØRMAT (16 F5.1)
3     FØRMAT ('␣', 10 F8.2)
      END
```

Observații :

1. Dimensiunile reale ale matricelor sînt citite la începutul programului. Asupra mărimilor nu se face în program nici o verificare, ele fiind presupuse corect transmise. În caz contrar rezultatul execuției programului ar fi eronat.

2. Dacă dimensiunile reale ale matricelor coincid cu cele din declarațiile corespunzătoare atunci listele de intrare/ieșire sînt mult mai simple :

```
READ (105, 1) A
READ (105, 1) B
WRITE (108, 3) C
```

3. Pe imprimantă identificarea liniilor matricei C, dacă numărul exact al coloanelor este diferit de 10, va fi dificilă. O formă îmbunătățită de afișare ar fi fost obținută astfel :

```
   DØ 22 I = 1, N
       WRITE (108, 3) (C(I, J), J = 1, P)
22 CØNTINUE
```

Dintre regulile care trebuie respectate cînd folosim această instrucțiune, le amintim pe următoarele :

— Valorile variabilelor *contor, vinit, vfin,* sau *vpas* nu pot fi schimbate prin instrucțiunile din corpul ciclului ;

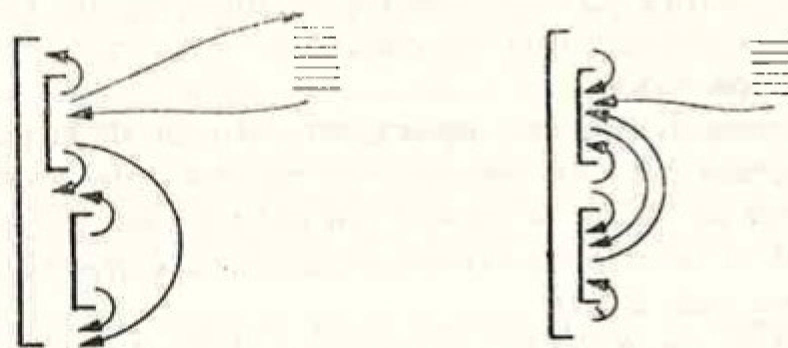


Fig. II.16

— Se poate sări din interiorul corpului în afara acestuia, dar nu și invers (vezi figura II.16). Saltul la o instrucțiune din interiorul corpului ciclului este permis numai dacă anterior s-a executat un salt din interiorul acestuia și nu s-au modificat parametrii ciclului :

— Ultima instrucțiune a corpului unui ciclu DØ nu poate fi : GØ TØ, IF aritmetic, PAUSE, STØP, RETURN, DØ, sau IF logic în care apare una din instrucțiunile de mai sus.

II.3.2. Subprograme

Cînd este necesară îndeplinirea de mai multe ori într-un program a acelorași calcule pentru date diferite, se poate folosi conceptul de *subprogram*. În *subprogram*, care poate fi privit ca o unitate independentă, *calculele ce se repetă sînt descrise o singură dată, iar din altă unitate de program se poate cere efectuarea acestora ori de cîte ori este nevoie.*

Atît subprogramele cît și programul principal constituie părți **componente** *ale aceluiași program. Totuși ele sînt independente, în sensul că fiecare dintre acestea poate fi alcătuit separat de celelalte, poate folosi nume de variabile care au mai fost întrebuințate și în restul programului și poate fi compilat și testat separat.*

Aceste caracteristici permit ca *la rezolvarea unei probleme să poată lucra mai multe persoane*, fiecare elaborînd cîte o parte din program. Unirea acestor părți pentru a rezolva problema considerată se realizează numai atunci cînd fiecare din ele este corectă din punct de vedere sintactic.

Important pentru cel ce folosește subprogramul este să știe care sînt datele de intrare și unde va fi găsit rezultatul. La orice centru de calcul există o așa-numită „bibliotecă", ce cuprinde asemenea programe.

În continuare vom numi **unitate de program** *fie programul principal, fie un subprogram. O unitate de program este o parte de sine stătătoare a unui program, independentă de celelalte unități de program (în sensul precizat mai sus).*

Orice unitate de program se alcătuiește după regulile corespunzătoare limbajului. Pînă acum toate programele pe care le-am scris au fost programe principale.

Ordinea instrucțiunilor FORTRAN într-o unitate de program este următoarea :

1. O instrucțiune SUBRØUTINE sau FUNCTIØN, dacă există.
2. Instrucțiunea IMPLICIT, dacă e cazul.
3. Declarații asupra datelor folosite în unitatea de program.
4. Cel puțin o instrucțiune executabilă.
5. Instrucțiunea END.

Dacă în unitatea de program apar instrucțiunile de la punctul 1 avem de-a face cu un program, în caz contrar, cu un *program principal.*

Deci din punct de vedere al formei, un subprogram este un grup de instrucțiuni FORTRAN a căror primă instrucțiune este SUBRØUTINE sau FUNCTIØN, iar ultima este END.

În mod similar, *un program principal e alcătuit dintr-un grup de instrucțiuni FORTRAN printre care nu figurează SUBRØUTINE, FUNCTIØN, dar a cărui ultimă instrucțiune este END.*

Din punct de vedere al utilizării, subprogramele se deosebesc prin :

a) *modul de alcătuire al unității de program respective ;*

b) *modul de referire (chemare)*, adică modul cum se exprimă într-o unitate de program necesitatea prelucrării efectuate de un subprogram ;

c) *modul de punere la dispoziția unității de program care cheamă subprogramul, a rezultatelor prelucrării efectuate de subprogram.*

Pe baza acestor caracteristici subprogramele se împart în :

— *subprograme de tip funcție* (numite și funcții externe) a căror primă instrucțiune este FUNCTIØN ;

— *subprograme de tip subrutină* (mai scurt, subrutine) a căror primă instrucțiune este SUBRØUTINE.

II.3.2.1. Funcții externe

Dacă prelucrarea cerută de găsirea valorii unei funcții nu se poate exprima printr-o singură expresie aritmetică, iar valoarea acestei expresii este necesară și în altă unitate de program, atunci pentru definirea funcției va fi alcătuită o unitate separată de program. În alte unități de program se pot face referiri la funcția astfel definită, avînd drept urmare calculul valorii funcției pentru diferite valori ale argumentelor ei.

Structura generală a unei funcții externe este cea din figura II.17.

tip FUNCTION(argumente fictive)
— *Declarații* — *Instrucțiuni executabile (cel puțin una)* — *Instrucțiunea RETURN*
END

Fig. II.17

În definirea funcțiilor externe apar două instrucțiuni noi, FUNCTIØN și RETURN.

Instrucţiunea FUNCTIØN

Forma generală :

$||$ t FUNCTIØN $f(a_1, a_2, \ldots, a_n)$

unde : t — Tipul numelui funcţiei (INTEGER, REAL, DØUBLE PRECISIØN, CØMPLEX, LØGICAL). Dacă este omis, tipul numelui funcţiei se stabileşte ca pentru o variabilă FORTRAN oarecare.

f — Numele funcţiei. El va păstra valoarea funcţiei.

a_1 $a_2, \ldots, a_n$ — Lista argumentelor fictive. Ea trebuie să conţină cel puţin un element. Elementele listei trebuie să fie distincte şi pot fi nume de variabile sau nume de tablouri, dar nu elemente ale unor tablouri.

Semnificaţie. Defineşte numele unei funcţii, tipul şi argumentele fictive ale acesteia.

Exemplu :

```
REAL FUNCTIØN MARE (PAR)
CØMPLEX FUNCTIØN CRIC (A, B)
FUNCTIØN MIC (X, Y, Z).
```

În exemplul al treilea funcţia MIC va fi de tip întreg, dacă o declaraţie IMPLICIT nu indică alt tip.

Instrucţiunea RETURN

Forma generală :

$||$ RETURN

Semnificaţie : Produce transferul controlului, în programul care a cerut execuţia funcţiei externe.

Într-o funcţie externă pot exista mai multe instrucţiuni RETURN.

La definirea unei funcţii externe trebuie respectate următoarele reguli :

R1. *Se va stabili care e prelucrarea îndeplinită de funcţie, numele funcţiei şi al argumentelor sale.*

R2. *Nu se pot folosi instrucţiunile SUBRØUTINE sau o altă instrucţiune FUNCTIØN.*

R3. *Numelui funcţiei trebuie să-i fie atribuită o valoare în cadrul subprogramului.*

Exemplul 1

Într-un program se repetă de mai multe ori calculul funcţiei :

$$y = \begin{cases} \dfrac{x}{1 + e^{\frac{1}{x}}} & \text{dacă } x < -2, \\ e^{-x^2} & \text{dacă } -2 \leqslant x < 4, \\ x^2 \sin \dfrac{1}{x} & \text{dacă } 4 \leqslant x. \end{cases}$$

S-a convenit să se alcătuiască funcţia externă YFUN folosind drept argument fictiv variabila X.

Respectînd regulile de mai sus, obținem subprogramul:

```
      REAL FUNCTIØN YFUN(X)
      IF(X.GE.—2.0) GØ TØ 31
      YFUN = X/(1.+EXP(1./X))
      RETURN
31    IF(X.GE.4) GØ TØ 32
      YFUN = EXP (—(X*X))
      RETURN
32    YFUN = X**2*SIN (1./X)
      RETURN
      END
```

Valoarea funcției va fi păstrată de YFUN, numele funcției. Se observă întrebuințarea a trei instrucțiuni RETURN.

Folosirea funcțiilor externe

Pentru utilizarea corectă a unei funcții externe, se va ține seama de următoarele reguli:

S1. *Se vor stabili care sînt argumentele efective pentru care se va calcula valoarea funcției.*

S2. *Argumentelor efective trebuie să le corespundă ca număr, ordine și tip argumentele fictive.*

— *Dacă argumentul fictiv corespunzător este un nume de variabilă atunci argumentul efectiv poate fi orice expresie de același tip.*

— *Dacă argumentul fictiv corespunzător este un tablou, trebuie ca argumentul efectiv corespunzător să fie tot un nume de tablou. În plus, pentru a evita orice erori este bine ca tablourile corespunzătoare să aibă aceleași dimensiuni.*

S3. *Găsirea valorii unei funcții externe pentru argumentele efective stabilite este cerută scriind numele funcției urmat între paranteze de lista argumentelor efective.*

S4. *Cînd este întîlnită o referire la o funcție sînt îndeplinite următoarele acțiuni:*

— *Se găsește valoarea fiecărui argument efectiv.*

— *Folosind aceste valori drept valori ale argumentelor fizice corespondente, se execută prelucrarea indicată de funcția externă.*

— *La întîlnirea primei instrucțiuni RETURN se revine în locul unde a fost făcută referirea la funcție. În acest moment la locul referirii se va pune valoarea numelui funcției, continuîndu-se execuția programului.*

Exemplul 2: Fie o referire la funcția YFUN (din exemplul 1) de forma:

Z = 2.4 + 5*YFUN (0.0).

După executarea acestei instrucțiuni, Z va avea valoarea 7.4 (adică 2.4 + 5).

Tablouri din subprograme ale căror dimensiuni sînt precizate în momentul execuției.

Funcțiile externe sînt utile în multe aplicații cu tablouri. Totuși de multe ori nu se cunosc dimensiunile exacte ale tablourilor ce sînt folosite, ci numai cîte dimensiuni au ele.

În subprograme (și numai în ele) *se poate utiliza o formă particulară a declarației* DIMENSIØN *în care nu se indică mărimea dimensiunilor unui tablou prin constante, ci prin variabile întregi, care sînt parametri ai subprogramului. Mărimile acestor dimensiuni vor fi precizate în momentul execuției.*

Exemplul 3 : Să se alcătuiască programul de tip funcție, PRØD, care să calculeze $\sum_{i=1}^{n} a_i b_i$ unde : a_i și b_i sînt componentele a două tablouri reale cu cîte n elemente.

Drept argumente fictive vom folosi :

A, B, — tablouri cu n elemente ;

N — variabila a cărei valoare va indica numărul elementelor tablourilor.

```
      FUNCTIØN PRØD (A, B, N)
      DIMENSIØN A(N), B(N)
      PRØD = 0
      DØ 77 K = 1, N
         PRØD = PRØD + A(K)*B(K)
77    CØNTINUE
      RETURN
      END
```

Iată și o referire la această funcție externă :

```
      REAL X(200), Y(200)
      . . . . . . . . . . .
      . . . . . . . . . . .
      . . . . . . . . . . .
      CALCUL = PRØD (X, Y, 200) + 5.72
      . . . . . . . . . . . . . . . . . . .
```

Cînd se întîlnește PRØD (X, Y, 200), la evaluarea funcției se consideră că dimensiunea celor două tablouri este 200 (valoarea argumentului efectiv corespunzător lui N).

II.3.2.2. Subrutine

Ca și funcțiile externe, subrutinele constituie unități distincte de program. Pentru alcătuirea subrutinelor sînt folosite cîteva instrucțiuni specifice, care vor fi studiate în continuare.

Instrucțiunea SUBRØUTINE

Servește la definirea subrutinelor, fiind prima instrucțiune a lor.

Forma generală :

| | SUBRØUTINE $s(a_1, a_2, \ldots, a_n)$

unde :

s — Numele subrutinei (se formează în mod similar cu numele funcțiilor).

$a_1, a_2, \ldots, a_n$ — Lista argumentelor fictive. Ea poate fi și vidă (în acest caz parantezele nu se mai scriu). Elementele listei sînt nume de variabilă sau de tablou (nu elemente de tablou).

Semnificație : Indică numele subrutinei, numărul, ordinea și tipul argumentelor. Spre deosebire de funcțiile externe, numelui subrutinei nu i se asociază un anumit tip. El servește numai pentru identificarea acesteia.

Instrucțiunea CALL

Forma generală :

| | CALL $s(p_1, p_2, \ldots, p_n)$

unde :

s — Numele subrutinei ;

$p_1, p_2, \ldots, p_n$ — Lista argumentelor efective.

Semnificație : Prin intermediul acestei instrucțiuni se cere **execuția subrutinei al cărei nume este** *s. Argumentele efective sînt indicate în listă și ele trebuie să coincidă ca număr, ordine și tip cu argumentele fictive din definiția subrutinei.*

Exemplu: O secvenţă de apelare a subrutinei MPR într-un program principal este:

```
      REAL A(20, 10), B(10, 30), C(20, 30)
      READ (105, 1) A, B
1     FØRMAT (10F8.5)
      CALL MPR (A, B, C, 20, 10, 30)
```

În acest moment matricea C va conţine produsul A·B.

II.3.3. Exerciţii

1. În figura II.18 este reprezentată structura unor cicluri DØ; arătaţi care din salturile indicate sînt permise şi care nu. De ce?

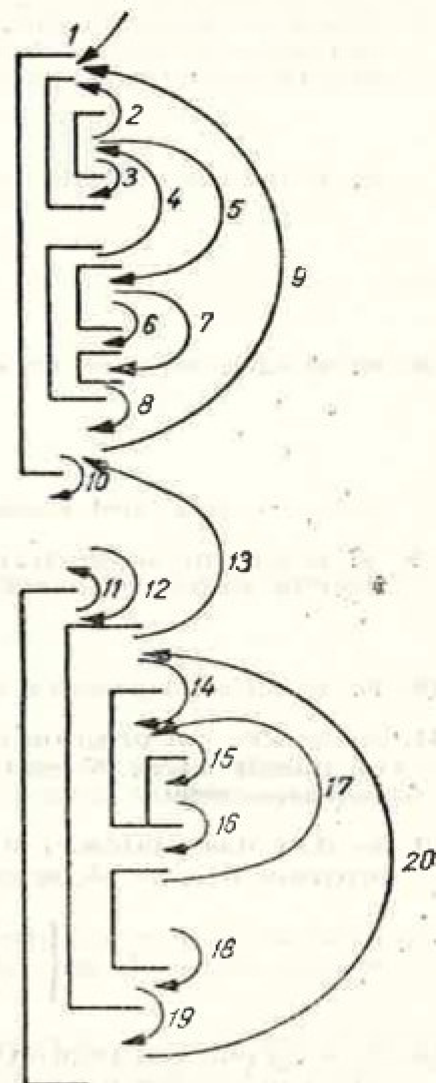


Fig. II.18

2. Identificaţi instrucţiunile corecte. De ce sînt greşite celelalte?

a) DØ 55 K = 1, 2, 3

b) DØ 38 VAX = K(L), K(2), K(3)

c) DØ 54 L(I) = 4, 5

d) DØ K = 1, 32, 4

e) DØ 77 K = 1, 4, 5, 6

f) DØ 88 1, 2, 3

3. Sînt corecte următoarele secvenţe? Să se justifice.

a)

```
      DØ 77 J = 1, 100
77    PLØG = PLØG - (-1)**J/J
```

b)

```
      S = 0
      DØ 33 K = 1, 10
33    S = S + K
```

c)

```
      DØ 88 L = K, L, M
      S(L) = 3*(M - L)
82    GØ TØ 60
```

d)

```
      DØ 11 K = 1, 100
11    A = 5
```

e)

```
      S = 0
      DØ 22 I = 1, M
      DØ 22 J = 1, N
22    S = S + A(1, J)
```

f)

```
      DØ 55 K = 1, 11, 3
      A(K) = B(K)
55    DØ 44 L = 1, 31
44    B(L) = 0
```

g)

```
      S = 0
      DØ 66 K = 1, M
      S = S + A(K)
66    A(K) = 10
```

h)

```
      S = 0
      DØ 99 K = 1,55
100   DØ 99 L = 1, 63
99    S = S*A(K, L)
      GØ TØ 100
```

Exemplu : O secvenţă de apelare a subrutinei MPR într-un program principal este:

```
      REAL A(20, 10), B(10, 30), C(20, 30)
      READ (105, 1) A, B
1     FØRMAT (10F8.5)
      CALL MPR (A, B, C, 20, 10, 30)
```

În acest moment matricea C va conţine produsul A·B.

II.3.3. Exerciţii

1. În figura II.18 este reprezentată structura unor cicluri DØ; arătaţi care din salturile indicate sînt permise şi care nu. De ce?

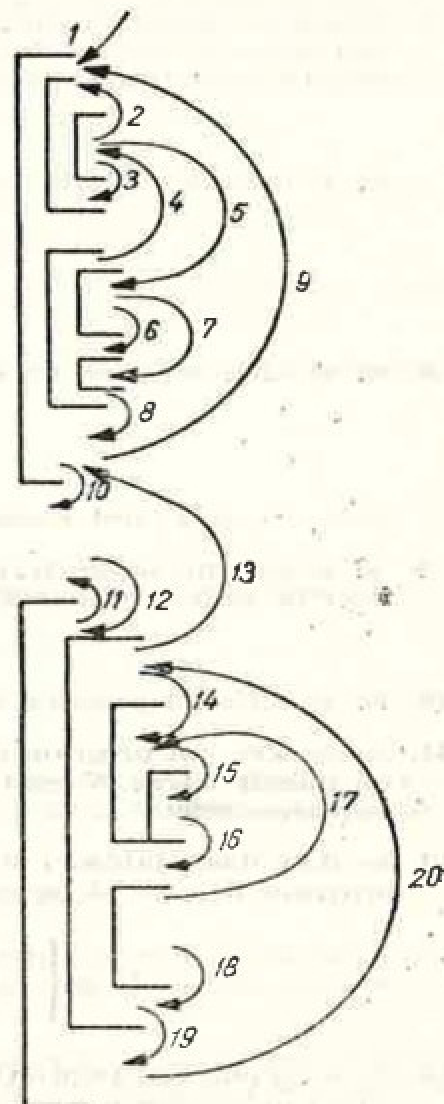


Fig. II.18

2. Identificaţi instrucţiunile corecte. De ce sînt greşite celelalte?

a) DØ 55 K = 1, 2, 3

b) DØ 38 VAX = K(L), K(2), K(3)

c) DØ 54 L(I) = 4, 5

d) DØ K = 1, 32, 4

e) DØ 77 K = 1, 4, 5, 6

f) DØ 88 1, 2, 3

3. Sînt corecte următoarele secvenţe? Să se justifice.

a)

```
      DØ 77 J = 1, 100
77    PLØG = PLØG – (–1)**J/J
```

b)

```
      S = 0
      DØ 33 K = 1, 10
33    S = S + K
```

c)

```
      DØ 88 L = K, L, M
      S(L) = 3*(M – L)
82    GØ TØ 60
```

d)

```
      DØ 11 K = 1, 100
11    A = 5
```

e)

```
      S = 0
      DØ 22 I = 1, M
      DØ 22 J = 1, N
22    S = S + A(I, J)
```

f)

```
      DØ 55 K = 1, 11, 3
      A(K) = B(K)
55    DØ 44 L = 1, 31
44    B(L) = 0
```

g)

```
      S = 0
      DØ 66 K = 1, M
      S = S + A(K)
66    A(K) = 10
```

h)

```
      S = 0
      DØ 99 K = 1,55
100   DØ 99 L = 1, 63
99    S = S*A(K, L)
      GØ TØ 100
```

4. Să se scrie un subprogram care să găsească numerele prime cuprinse între 1 şi N.

5. Scrieţi un subprogram pentru găsirea numerelor perfecte mai mici ca N. (Un număr perfect este un număr care este egal cu suma divizorilor lui, excluzîndu-se el însuşi : $6 = 1 + 2 + 3$; $28 = 1 + 2 + 4 + 7 + 14$).

6. Să se scrie un program care să imprime primii 20 termeni ai şirului lui Fibonacci

$$a_1 = 1,\ a_2 = 1,\ \text{iar}\ a_i = a_{i-2} + a_{i-1}.$$

7. Scrieţi un program care să citească de pe cartele valorile a două tablouri A şi B unidimensionale, fiecare avînd cîte 25 elemente. Folosind elementele celor două tablouri se completează elementele unui al treilea tablou AB de aceeaşi dimensiune, după regula :

$$AB_i = A_i \cdot B_i.$$

Să se imprime valorile celor 3 tablouri şi valoarea calculată după formula :

$$\frac{\sum_{i=1}^{25} A_i\ B_i}{25}.$$

8. Să se scrie un program care să calculeze

$$\sum_{i=1}^{100} \sqrt{|a_i - b_i|\ |a_i + b_i|}$$

unde : a_i şi b_i sînt elementele a doi vectori reali A şi B cu cîte 100 de elemente.

9. Să se scrie un subprogram FORTRAN care să găsească toate numerele întregi x şi y mai mici în valoare absolută decît un N fixat, care sînt soluţii ale ecuaţiilor :

$$\text{a) } 3x + 5y = 11\ ;\quad \text{b) } 3x + 2y = 5.$$

10. Să se scrie un program care să alcătuiască tabla de adunare în baza 8.

11. Să se scrie un program care să calculeze 1 !, 2 !, 3 !,... În momentul cînd K ! va depăşi un număr întreg N fixat să nu se mai calculeze $(K + 1)$! şi o variabilă V să primească valoarea 9999.

12. Se dau două tablouri A şi B fiecare avînd 30 elemente. Se cere să se alcătuiască un program în care să fie completate elementele tabloului X cu 30 elemente, astfel :

$$X_i = \begin{cases} -1 \text{ dacă } a_i < b_i \\ \ \ 0 \text{ dacă } a_i = b_i \text{ pentru } i = 1, \ldots, 30. \\ +1 \text{ dacă } a_i > b_i. \end{cases}$$

13. Pe o cartelă sînt imprimate 50 de cifre zecimale. Să se scrie un program care să găsească numărul cifrelor pare, diferite de zero, al celor impare precum şi al celor egale cu zero.

14. Pe cartele sînt imprimate numere întregi fără semn, diferite de zero, fiecare avînd cel mult 9 cifre. Să se alcătuiască un program care să afişeze secvenţele întîlnite (o secvenţă e un grup de cel puţin 2 numere consecutive) şi numărul acestora.

Observaţie. O secvenţă va fi afişată indicînd valoarea iniţială urmată de numărul de elemente al secvenţei.

15. Cele 90 elemente reale ale unui tablou V sînt citite de pe cartele cu formatul 5F8.3. Să se alcătuiască un program astfel ca variabila K să indice cel mai mic număr de elemente ale tabloului pe care trebuie să le adunăm pentru ca suma să depăşească numărul 1 000. Dacă suma celor 90 de elemente nu depăşeşte valoarea 1 000 atunci K va primi valoarea -1.

16. Să se alcătuiască funcţia PVECT care va da produsul primelor k elemente ale vectorului A cu n componente.

17. Fie tabelul A cu N elemente. Construiţi funcţia externă PART care calculează suma elementelor tabloului pentru care $|A(I)| > R$, unde valoarea lui R e precizată.

18. Să se alcătuiască funcţia MEDIE care va calcula pentru un vector real dat A cu N elemente şi un K întreg fixat (K ≤ N) expresia

$$\frac{A(1) + \ldots + A(K)}{K}.$$

19. Să se alcătuiască o funcţie care să verifice dacă un tablou A cu N linii şi N coloane este simetric. (Indicaţie: Rezultatul se va comunica prin valori diferite ale funcţiei pentru cazurile posibile).

20. Să se construiască o funcţie care să calculeze pentru un tablou X cu M linii şi N coloane suma

$$\sum_{i=1}^{M} \sum_{j=1}^{N} |x_{ij}|^2.$$

21. Scrieţi o funcţie care să găsească rădăcina cea mai mare în valoare absolută a unei ecuaţii de gradul doi.

22. Să se alcătuiască o subrutină care pentru un tablou A cu 15 linii şi 30 coloane să calculeze

$$\frac{\sum_{i=1}^{15} \sum_{j=1}^{30} A(I, J)}{450}.$$

23. Fiind dat tabloul întreg TAB cu M linii şi N coloane alcătuiţi o subrutină care să calculeze:

— suma elementelor pozitive;
— suma elementelor negative;
— numărul elementelor egale cu zero.

24. Este posibil ca într-o subrutină cu un singur argument, să se transmită dintr-un program o valoare cu care să se efectueze prelucrarea şi apoi să se aducă în programul iniţial rezultatul prelucrării?

25. Precizaţi care sînt deosebirile dintre funcţiile externe şi subrutine.

26. Să se construiască o funcţie care să arate numărul de apariţii ale valorii *val* între poziţiile p_i şi p_{sf} ale tabloului A, unidimensional cu n elemente. Să se testeze funcţia.

Observaţie. În program se va verifica corectitudinea argumentelor; dacă *val* nu poate fi calculată, un indicator de eroare va fi poziţionat cu o valoare, iar dacă se poate calcula va primi o altă valoare.

De exemplu:

$$IER = \begin{cases} -2 \text{ pentru caz corect} \\ -1 \text{ pentru caz incorect} \end{cases}$$

27. Citiţi un şir de caractere şi afişaţi la imprimantă şirul în ordine inversă. Construiţi o subrutină care să inverseze un şir de caractere.

28. Scrieţi un program care să adune 2 numere romane.

29. Să se scrie o subrutină care să găsească valoarea maximă a elementelor unui tablou cu n elemente şi poziţia pe care se află primul element egal cu această valoare. Să se dea exemplu de folosire a subrutinei.

30. Scrieţi un subprogram care să afişeze sub formă de text valoarea unui număr întreg pozitiv, cu cel mult trei cifre.

REZOLVARI

Paragraful I.2

Rezolvarea problemei 1 a)

a	v.s.	val. întregi
b	v.s.	val. întregi

citeşte a

citeşte b

scrie b

scrie a

stop

Rezolvarea problemei 2 a)

x	v.s.	val. reale
f	v.s.	val. reale

citeşte x

atribuie $f \leftarrow exp(-x)*sin(x)$

scrie f

stop

Rezolvarea problemei 2 b)

x	v.s.	val. reale
y	v.s.	val. reale
g	v.s.	val. reale

citeşte x

citeşte y

atribuie $g \leftarrow \sqrt[3]{x^3 + y^3}$

scrie g

stop

Rezolvarea problemei 2 c)

x	v.s.	val. reale
y	v.s.	val. reale
fi	v.s.	val. reale
psi	v.s.	val. reale

citeşte x

citeşte y

atribuie $fi \leftarrow x^5*\cos\ y^2$

atribuie $psi \leftarrow y^3 + ln(x^2 + 1)$

scrie *fi*

scrie *psi*

stop

Rezolvarea problemei 3

unghi	v.s.	val. reale
t	v.s.	val. reale
grade	v.s.	val. întregi
min	v.s.	val. întregi
sec	v.s.	val. întregi

citeşte *unghi*

atribuie $t \leftarrow 180*unghi/3{,}14159$

& $grade \leftarrow [t]$

& $t \leftarrow (t - grade)*60$

& $min \leftarrow [t]$

& $t \leftarrow (t - min)*60$

& $sec \leftarrow [t]$

scrie *grade, min, sec*

stop

Observaţie. S-a notat cu $[t]$ partea întreagă a lui t.

Rezolvarea problemei 1 *a)*

m	v.s.	val. întregi
n	v.s.	val. întregi

```
citește m, n
dacă m > n atunci
    scrie m, n
altfel
    scrie n, m
■
stop
```

Rezolvarea problemei 4

x	v.s.	val. reale
f	v.s.	val. reale

```
citește x
dacă x < – 1 atunci
    atribuie f ← 1 – x
    altfel
  dacă x ⩽ 1 atunci
    atribuie f ← √(1 – x²)
  altfel
    atribuie f ← x – 1
  ■
■
scrie f
stop
```

Rezolvarea problemei 5

ap	v.s.	v. întregi poz.
lp	v.s.	v. întregi poz.
zp	v.s.	v. întregi poz.
an	v.s.	v. întregi poz.
ln	v.s.	v. întregi poz.
zn	v.s.	v. întregi poz.
vîrstă	v.s.	v. întregi poz.

```
citește ap, lp, zp, an, ln, zn
dacă ap < an atunci
  scrie „date eronate"
altfel
  atribuie vîrsta ← ap – an
  dacă ln > lp ∨ (ln = lp ∧ zn > zp) atunci
        atribuie vîrsta ← vîrsta – 1
  ■
■
scrie vîrsta
stop
```

Rezolvarea problemei 7

a) Dacă notăm cu α_0 şi β_0 valorile citite în m şi n, iar cu α_1 şi β_1, α_2 şi $\beta_2, \ldots$ valorile din m şi n la sfîrşitul execuţiilor succesive ale secvenţei din ciclu, avem relaţiile:

$$|\alpha_0 - \beta_0| > |\alpha_1 - \beta_1| > \ldots > |\alpha_i - \beta_i| > \ldots \geqslant 0.$$

Deci:

$$(\exists n)(n \in \mathbf{N}) \, |\alpha_n - \beta_n| = 0 \Rightarrow \alpha_n = \beta_n.$$

Aceasta este relaţia care ne asigură ieşirea din ciclu.
Ştiind că c.m.m.d.c. (a, b) = c.m.m.d.c. $(a - b, b)$ se poate arăta prin inducţie că:

$$\text{c.m.m.d.c. } (\alpha_i, \beta_i) = \text{c.m.m.d.c. } (\alpha_0, \beta_0) = \gamma.$$

Într-adevăr, pentru $i = 0$ egalitatea este evidentă.
Să presupunem că: c.m.m.d.c. $(\alpha_j, \beta_j) = \gamma$ oricare ar fi $j \leqslant k$ şi fie cazul $i = k + 1$. Vom avea

— fie $\alpha_{k+1} = \alpha_k - \beta_k$ şi $\beta_{k+1} = \beta_k$

— fie $\alpha_{k+1} = \alpha_k$ şi $\beta_{k+1} = \beta_k - \alpha_k$.

În ambele cazuri c.m.m.d.c. $(\alpha_{k+1}, \beta_{k+1}) = \gamma$.

b)

m	v.s.	v. întregi strict poz.
n	v.s.	v. întregi strict poz.
gata	v.s.	v. logice

```
citeşte m, n
atribuie gata ← fals
┌ciclează
│ ┌dacă m = n atunci
│ │   atribuie gata ← adevărat
│ │ altfel
│ │ ┌dacă m > n atunci
│ │ │ atribuie m ← m − n
│ │ │ altfel
│ │ │ atribuie n ← n − m
│ │ └■
│ └■
│ pînă gata
└■
  scrie m
  stop
```

Rezolvarea problemei 8

n	v.s.	v. întregi poz.
i	v.s.	v. întregi poz.
f	v.s.	v. întregi poz.

```
citeşte n
atribuie i ← 1 & f ← 1
┌cît timp 1 < i ≤ n repetă
│     atribuie f ← f*i
└■
scrie f
stop
```

Discuţie

1° Factorialul este o funcţie cu o creştere rapidă. Dacă $5! = 120$, în schimb $10! = 3628800$, iar $1000!$ este de ordinul lui 10^{2500}. De aceea calculele al căror număr de paşi se exprimă cu ajutorul factorialului trebuie tratate cu cea mai mare atenţie deoarece timpul lor de execuţie poate creşte rapid în funcţie de datele de intrare.

2° Formula lui Stirling ne dă o alternativă simplă de calcul cu aproximaţie al factorialului:

$$n! \approx \sqrt{2 \cdot \pi \cdot n} \left(\frac{n}{e}\right)^n$$

Eroare relativă: circa $\dfrac{1}{12 \cdot n}$.

Rezolvarea problemei 9

Intrare : valoarea (reală) a erorii absolute maxim admisibile.

Ieşire : valoarea (reală) a soluţiei aproximative.

Zona datelor : *eps* conţine eroarea absolută maxim admisibilă ;

a, b conţin coordonatele (pe axa reală) capetelor intervalului la un moment dat ;

c conţine coordonata mijlocului intervalului ;

gata conţine o valoare logică ce indică prin valoarea **adevărat** atingerea (puţin probabilă) a soluţiei exacte ;

f conţine valoarea funcţiei $f(x) = x \cdot e^x - 1$ într-un punct dat.

eps	v.s.	val. reale strict poz.
a	v.s.	val. reale strict poz.
b	v.s.	val. reale strict poz.
c	v.s.	val. reale strict poz.
gata	v.s.	val. logice
f	v.s.	val. reale

```
citeşte eps
atribuie a ← 0 & b ← 1 & gata ← fals
┌ciclează
│  atribuie c ← (a + b)/2
│  atribuie f ← c*exp(c) − 1
│ ┌dacă f = 0 atunci
│ │    atribuie gata ← adevărat
│ │ altfel
│ │ ┌dacă f < 0 atunci
│ │ │    atribuie a ← c
│ │ │ altfel
│ │ │    atribuie b ← c
│ │ └■
│ └■
│  pînă |a − b| < eps ∨ gata
└■
scrie c
stop
```

Rezolvarea problemei 10

Intrare : — valoarea ordinului (p),

— valoarea din care se extrage radicalul (a),

— valoarea iniţială (x_0),

— eroarea maxim admisibilă (ε).

Ieşire : valoarea aproximativă a rădăcinii.

Numele celulelor din memorie utilizate respectă notațiile din indicație.

p	v.s.	val. întregi poz.
a	v.s.	val. reale
xzero	v.s.	val. reale
eps	v.s.	val. reale poz.
x	v.s.	val. reale
er	v.s.	val. reale

```
citeşte p, a, x zero, eps
    atribuie x ← ((p − 1)*xzero + a/xzero^(p−1))/p
┌dacă x ≠ xzero atunci
│   atribuie er ← |x|
│ ┌cît timp er < eps repetă
│ │ ┌atribuie er ← er*(p − 1)*(1 − a/x^p)/p
│ │ │   & x ← ((p − 1)*x + a/x^(p−1))/p
│ │ └■
│ └■
└■
scrie x
stop
```

Rezolvarea problemei 11

a)

n	var. s	v. întregi poz.
v	tabl. 1 dim (1000)	v. reale
i	var. s.	v. întregi poz.
max	var. s	v. reale

```
citeşte n
citeşte (v(i), i = 1, n)
atribuie max ← v(1) & i ← 1
┌cît timp i ≤ n repetă
│         ┌dacă max < v(i) atunci
│         │ atribuie max ← v(i)
│         └■
│     atribuie i ← i + 1
└■
scrie max
stop
```

b)

n	var. s.	v. întregi poz.
v	var. s.	v. reale
i	var. s.	v. întregi poz.
max	var. s.	v. reale

```
citeşte n
citeşte v
atribuie max ← v & i ← 1
┌cît timp i ≤ n repetă
│         ┌dacă max < v atunci
│         │ atribuie max ← v
│         └■
│           atribuie i ← i + 1
└■
scrie max
stop
```

Observaţie. Programul de la pct. b) necesită o zonă minimă de date deoarece nu memorează tot şirul dintr-o dată ci citeşte de pe banda de intrare de îndată ce are nevoie de o nouă valoare. Soluţia este posibilă, deoarece valorile citite sînt necesare o singură dată, după care ele se pot şterge. Situaţia diferă de cea întîlnită la algoritmul de ordonare.

Rezolvarea problemei 13

c)

m	var. s.	v. întregi poz.
nr	var. s.	v. reale
prod	var. s.	v. reale
i	var. s.	v. întregi poz.

```
citeşte m
atribuie prod ← 1 & i ← 1
┌ciclează
│          citeşte nr
│          ┌dacă nr≠0 atunci
│          │    atribuie prod ← prod*nr
│          └■
│            atribuie i ← i + 1
│  pînă i > m
└■
scrie prod
stop
```

h₁)

n	var. s.	v. întregi poz.
nr	tab. 1 dim	v. reale
suma	var. s.	v. reale
i	var. s.	v. întregi poz.
medie	var. s.	v. reale
sumap	var. s.	v. reale
disp	var. s.	v. reale
abmp	var. s.	v. reale

```
citeşte n
citeşte (nr(i), i = 1, n)
atribuie suma ← 0 & i ← 1
┌cît timp i ≤ n repetă
│       ┌atribuie suma ← suma + nr(i)
│       │  & i ← i + 1
│       └■
└■
atribuie medie ← suma/n
atribuie sumap ← 0 & i ← 1
┌cît timp i ≤ n repetă
│    ┌atribuie sumap ← sumap + (nr(i) − medie)²
│    │    & i ← i + 1
│    └■
└■
┌atribuie disp ← sumap/n
│      & abmp ← √disp
└■
scrie disp, abmp
stop
```

h₂)

n	var. s.	v. întregi poz.
nr	var. s.	v. reale
suma	var. s.	v. reale
i	var. s.	v. întregi poz.
sumap	var. s.	v. întregi poz.
disp	var. s.	v. întregi poz.
abmp	var. s.	v. întregi poz.

```
citeşte n
citeşte nr
atribuie suma ← nr & sumap ← nr²
atribuie i ← 2
┌cît timp i ≤ n repetă
│       citeşte nr
│       atribuie suma ← suma + nr
│       atribuie sumap ← sumap + nr²
│       atribuie i ← i + 1
└■
atribuie disp ← sumap/n − (suma/n)²
atribuie abmp ← √disp
scrie disp, abmp
   stop
```

Observaţie. Comparînd cei doi algoritmi de la pct. h (h_1 şi h_2) constatăm că h_2 foloseşte mai eficient memoria şi timpul de execuţie, deoarece valorile nu se memorează toate deodată, prelucrarea realizîndu-se valoare cu valoare. Acest lucru a fost posibil printr-o transformare a relaţiei ce dă dispersia astfel:

$$D = \frac{\sum_{i=1}^{n}(x_i - m)^2}{n} = \frac{\sum_{i=1}^{n}(x_i^2 - 2x_i \cdot m + m^2)}{n} = \frac{\sum_{i=1}^{n} x_i^2}{n} - 2 \cdot \frac{\sum_{i=1}^{n} x_i}{n} \cdot m +$$

$$+ \frac{\sum_{i=1}^{n} m^2}{n} = \frac{\sum_{i=1}^{n} x_i^2}{n} - 2m^2 + m^2 = \frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2.$$

Rezolvarea problemei **14 b**

În afara notaţiilor făcute pentru programele din figura I.27 mai utilizăm: k — nume al celulei care memorează rangul ultimei valori ce a trebuit să fie modificată în timpul căutării: se iniţializează cu 0 şi constituie astfel şi un „semafor" pentru ieşirea din ciclu, lim — nume al celulei ce memorează între diferitele parcurgeri ale şirului rangul ultimei inversări.

n	var. simplă	v. întregi [1, 100]
a	tab. 1 dim. (100)	v. reale
i	var. simplă	v. întregi
t	var. simplă	v. reale
k	var. simplă	v. întregi poz.
lim	var. simplă	v. întregi poz.
m	var. simplă	v. întregi

```
citeşte n
citeşte (a(i), i = 1, n)
atribuie lim ← n
┌ciclează
│   atribuie k ← 0&m ← lim − 1
│   ┌pentru i = 1, m execută
│   │  ┌dacă a(i) < a(i + 1) atunci
│   │  │      atribuie t ← a(i)&a(i) ← a(i + 1)&a(i + 1) ← t
│   │  │     atribuie k ← i
│   │  └■
│   └■
│   atribuie lim ← k
│ pînă k = 0
└■
scrie (a(i), i = 1, n)
stop
```

Rezolvarea problemei **14 c**

Folosim majoritatea notaţiilor algoritmului din figura I.27. Celelalte rezultă imediat din context.

```
citeşte n
citeşte (a(i), i = 1, n)
┌atribuie m ← n − 1
│pentru i = 1, m execută
│      atribuie p ← i + 1
│     │pentru j = p, n execută
│     │  ┌dacă a(i) < a(j) atunci
│     │  │  atribuie t ← a(i)&a(i) ← a(j)&a(j) ← t
│     │  └■
│     └■
└■
scrie (a(i), i = 1, n)
stop
```

Rezolvarea problemei 16

n	var. s.	v. întregi poz.
a	tabl., 2 dim. (20, 20)	v. reale
b	tabl., 1 dim. (20)	v. reale
x	tabl., 1 dim. (20)	v. reale
i	var. s.	v. întregi
j	var. s.	v. întregi
k	var. s.	v. întregi
s	var. s.	v. reale

```
citeşte n
citeşte ((a(i, j), j = 1, n), i = 1, n), (b(i), i = 1, n)
atribuie x(1) ← b(1)/a(1, 1)
┌pentru i = 2, n execută
│    atribuie s ← b(i)&k ← i − 1
│  ┌pentru j = 1, k execută
│  │     atribuie s ← s − a(i, j)*x(j)
│  └■
│    atribuie x(i) ← s/a(i, i)
└■
scrie (x(i), i = 1, n)
stop
```

Rezolvarea problemei 19

n	var. s.	v. întregi poz.
număr	var. s.	v. reale
nr	tabl., 2 dim. (1000, 2)	v. reale
k	var. s.	v. întregi poz.
j	var. s.	v. întregi poz.
i	var. s.	v. întregi poz.
găsit	var. s.	v. logice

```
citeşte n
citeşte număr
atribuie nr(1, 1) ← număr & nr(1, 2) ← 1 & k ← 1
┌pentru i = 2, n execută
│     citeşte număr
│     atribuie găsit ← fals
│     ┌pentru j = 1, k execută
│     │   ┌dacă număr = nr(j, 1) atunci
│     │   │     ┌atribuie nr(j, 2) ← nr(j, 2) + 1
│     │   │     │ & găsit ← adevărat
│     │   │     └■
│     │   └■
│     └■
│     ┌dacă nu găsit atunci
│     │     atribuie k ← k + 1 & nr(k, 1) ← număr & nr(k, 2) ← 1
│     └■
└■
scrie ((nr(i, j), j = 1, 2), i = 1, k)
stop
```

Rezolvarea problemei 20

număr	var. s.	v. întregi
i	var. s.	v. întregi
div	var. s.	v. logice

```
citeşte număr
atribuie i ← 2 & div ← fals
┌ciclează
│     ┌dacă număr = (număr/i)*i atunci
│     │        atribuie div ← adevărat
│     └■
│     atribuie i ← i + 1
│pînă div ∨ i > (număr/2)
└■
┌dacă div atunci
│        scrie „nu"
│altfel
│            scrie „da"
└■
stop
```

Paragraful II.1.7

3. a, b, e, h. **4.** 1259.4 ; −2.959 ; 4.0001E4 ; 1.E25 ; 1.0E−7 ; 1.0E−7. **6.** a, c, d. **7.** a, b, c, d, e. **8.** a, e. **11.** b, c, e, f. **15.** O soluţie este adăugarea unui I la începutul fiecărui nume. **19.** Sînt contradictorii. **21.** Trebuie declarat explicit.

25. a) (X + 1)**(N + 2) + X**(2*N + 1) b) X**2 − 2*X*COS(ALFA)+ 1 e) (1/B − − 1A)/((M/C) + A*B) i) − COS(ALFA)**4*SIN(ALFA)**3/(P + 1). **28.** a) 0.0 b) 3.28 c) 0.0 **30.** A ← 30.0, B ← 8.0, C ← 8.0, P ← 0D0, W ← (6.3, 7.65), U ← (3., −1.), L1 ← .FALSE., IA ← 2, A11 ← 2.0, A2 ← 0.0, T ← 10.0D0, I ← 2, I1 ← 4, A1 ← 0.0, M ← 9.

Paragraful II.2.10

1\. a)

```
   || IF(A.GE.B) GØTØ 11
   || X = − 21.5
   || GØ TØ 22
11 || X = 15.6
22 || CØNTINUE
```

b)

```
  || IF((ALFA+BETA).GT.0) GØ TØ 3
  ||    IF(X.LT.7) GØ TØ 2
  ||          Y = 121.1
  ||    GØ TØ 4
2 ||          Y = − 21
  ||    GØ TØ 4
3 ||          Y = 15.7
4 || CØNTINUE
```


2. a) IF(A.LT.0.AND.B.GT.0.ØR.ALFA.EQ.0) X = − 2

b)

```
      IF(X.GT.0.000001.AND.X.LE.1) GØ TØ 777
      STØP
```

3.

```
      IF(K − K/2*2.EQ.0) GØ TØ 615
```

5. a) 28.00 b) 11.8

7.

```
      INTEGER S
      READ (105, 10) N
      S = 0
      K = 1
1     S = S + K
      K = K + 1
      IF(K.LE.N) GØ TØ 1
      WRITE (108, 11) S
      STØP
10    FØRMAT (I3)
11    FØRMAT ('␣', I7)
      END
```

Paragraful II.3.3

1. Permise: 2, 3, 6, 8, 9, 10, 11, 16, 18, 19; Interzise: 1, 4, 5, 7, 12, 13, 14, 15, 17, 20

2. a

3. Corecte: a, b, d, e, g.

6.

```
      INTEGER A (20)
      A (1) = 1
      A (2) = 1
      DØ 1 K = 3, 20
1     A(K) = A(K − 2) + A(K − 1)
      WRITE (108, 11) A
      STØP
11    FØRMAT ('␣', 10I9)
      END
```

11.

```
      INTEGER FACT
      N = . . .
      FACT = 1
      DØ 2 K = 1, N
      FACT = FACT*K
      WRITE (108, 11) FACT
      IF (FACT. GT.N) GØ TØ 3
2     CØNTINUE
      . . . . . . . . . . . .
3     V = 9999
      . . . . . . . . . . . .
```

12.

```
      REAL A (30), B(30), X(30)
      . . . . . . . . . . . . . .
      DØ 1 I = 1, 30
      IF (A(I). LT. B(I)) X(I) = − 1
      IF (A(I).EQ.B (I)) X(I) = 0
1     IF (A(I).GT.B(I)) X(I) = 1
```

6.

```
      REAL FUNCTIØN PVECT (A, K)
      REAL A (N).
      PVECT = 1
      DØ 1 I = 1, K
1     PVECT = PVECT*A(I)
      RETURN
      END
```

20.

```
      FUNCTIØN PRØD (X, N, M)
      REAL X (M, N)
      PRØD = 1
      DØ 1 I = 1, M
      DO 1 J = 1, N
1     PRØD = PRØD*AMØD X (1, J)
     +**2
      RETURN
      END
```

CUPRINS

